



Mathematica для математиков.

Часть 4. Решение дифференциальных уравнений.

Многие прикладные задачи сводятся к решению обыкновенных дифференциальных уравнений (ОДУ) или их систем. Для некоторых уравнений и систем имеются формулы «точного» решения, и пакет *Mathematica* умеет их находить. В тех задачах, для которых символьное решение построить не удастся, *Mathematica* может находить численное решение. В решениях она использует богатый набор специальных функций и свои интерполяционные функции, представляя решение в форме, в которой они могут быть непосредственно использованы. Данная часть пособия знакомит Вас с основными функциями системы, предназначенными для символьного и численного решения ОДУ и их систем, а также с вспомогательными функциями, используемыми при их исследовании.

Перед вами открыта четвертая часть нашего пособия. Предыдущие три части

1. Основы алгебраических вычислений;
2. Графические возможности системы;
3. Реализация основных понятий математического анализа;

вы можете найти на сайте geometry@karazin.ua в разделе автора «Документы».

Оглавление

4.1 Символьные решения ОДУ и их систем.....	2
4.1.1 Решение обыкновенных дифференциальных уравнений.....	2
4.1.2 Символьное решение систем ОДУ	16
4.2 Численные решения	19
4.2.1 Численное решение ОДУ	19
4.2.2 Численное решение систем ОДУ	33
4.3 Представление решений ОДУ в системе <i>Mathematica</i>	39
4.4 Дифференциально – алгебраические уравнения.	46
4.5 Примеры исследования ОДУ	48
4.5.1 Движение материальной точки по прямой.	48
4.5.2 Движение упругого мяча	49
4.5.3 Равновесие струны под действием сосредоточенных сил.	50
4.5.4 Статический прогиб сети нитей	53
4.5.5 Прогиб балки.....	55

4.5.6 Расчет рамы	60
4.5.7 Поперечные колебания струны с грузами	62
4.5.8 Модель Ферми – Улама – Пасты.....	64
4.5.9 Движение тела, брошенного под углом к горизонту	68
4.5.10 Движение тел под действием тяготения	70
4.5.11 Математический маятник	74
4.5.12 Колебание массивного тела под действием пружин.....	76
4.5.13 Генератор пилообразных напряжений	78
4.5.14 Двухвидовая модель Вольтерра «хищник – жертва»	81
Замечания о выполнении примеров	83

4. Решение дифференциальных уравнений в пакете Mathematica.

4.1 Символьные решения ОДУ и их систем.

Система *Mathematica* обладает обширными возможностями решения обыкновенных дифференциальных уравнений и их систем в символьном виде. Для этого используется функция `DSolve`, в алгоритме которой реализовано большинство известных на сегодняшний день аналитических методов.

При поиске решения функция `DSolve` полагает, что имена искомых функций и имя независимой переменной не содержат значений в рабочем пространстве системы. Если это не так, то функция `DSolve` выдаст сообщение об ошибке. В таком случае имена переменных и функций, используемых `DSolve`, следует удалить из рабочего пространства командой `Remove[имя1, имя2, ...]`. Все последующие примеры предполагают, что это уже сделано. Если все же у вас возникнут проблемы с выполнением примеров, то обратитесь в конец пособия в раздел «Замечания о выполнении примеров», где мы кратко описываем возможные трудности, с которыми вы можете столкнуться, и способы их устранения.

4.1.1 Решение обыкновенных дифференциальных уравнений.

Система *Mathematica* позволяет решать многие ОДУ в символьном виде с помощью функции `DSolve`. Функция имеет следующий синтаксис

`DSolve[уравнение, y, x]`

Она определяет функцию y , считая x независимой переменной уравнения. Первый аргумент – уравнение (или список уравнений), записанное в терминах функций и ее производных ($y[x]$, $y'[x]$, $y''[x]$ и т.д.), но допустимо использовать другие способы обозначения производных (например, $D[y[x], x]$, $y^{(2)}[x]$, $\partial_x y[x]$, $\partial_{xx} y[x]$ и др.). Второй аргумент – имя искомой функции. Третий аргумент – независимая переменная. Результат возвращается в виде списка правил подстановки.

`DSolve[y'[x] == x^2, y[x], x]`

$\{ \{ y[x] \rightarrow \frac{x^3}{3} + C[1] \} \}$

Здесь $C[1]$ представляет произвольную константу интегрирования.

Функция `DSolve` в качестве первого аргумента может принимать список уравнений, среди которых могут быть и начальные условия.

DSolve[$\{D[y[x], x] == x^2, y[0] == 1\}, y[x], x]$
 $\{\{y[x] \rightarrow \frac{1}{3}(3 + x^3)\}\}$

Она может также находить решение с произвольными параметрами в качестве коэффициентов.

DSolve[$y'[x] == ay[x], y[x], x]$
 $\{\{y[x] \rightarrow e^{ax} C[1]\}\}$

Символьные параметры могут присутствовать в качестве коэффициентов или значений в начальных условиях.


DSolve[$\{y'[x] == ay[x], y[c] == b\}, y[x], x]$
 $\{\{y[x] \rightarrow be^{-ac+ax}\}\}$

Функция `DSolve` ищет решение в символьном виде и пытается вернуть решение в квадратурах. Если уравнение содержит неопределенную функцию, то в решении появляются неопределенные интегралы.

DSolve[$y'[x] == f[x], y[x], x]$
 $\{\{y[x] \rightarrow C[1] + \int_1^x f[K[1]] dK[1]\}\}$

Здесь $K[1]$ обозначает переменную интегрирования, а $C[1]$ представляет произвольную константу. Если такие обозначения вас не устраивают, то можно выполнить замену/подстановку следующим образом

DSolve[$y'[x] - y[x] == f[x], y[x], x\} /. \{K[1] \rightarrow t, C[1] \rightarrow C_1\}$
 $\{\{y[x] \rightarrow e^x \int_1^x e^{-t} f[t] dt + e^x C_1\}\}$

Для ввода C_1 наберите C , затем комбинацию `Ctrl-1`. В полученный шаблон  введите индекс 1. Для возврата на нормальный уровень ввода наберите комбинацию `Ctrl-пробел`. Запись `/. имя1->имя2` обозначает подстановку (символы `/.`) в предшествующее выражение вместо каждого вхождения набора символов **имя1** другого набора символов **имя2**.

Функция `DSolve` умеет работать с некоторыми разрывными функциями, например

DSolve[$D[y[x], \{x, 2\}] - y[x] == \text{UnitStep}[x], y[x], x]$
 $\{\{y[x] \rightarrow e^x C[1] + e^{-x} C[2] + \frac{1}{2} e^{-x} (-1 + e^x)^2 \text{UnitStep}[x]\}\}$

Здесь `UnitStep` – функция «единичная ступенька»

$$\text{UnitStep}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Однако с задачей Коши для ДУ с разрывными функциями *Mathematica* иногда не справляется

DSolve[$\{y'[x] == \text{UnitStep}[x], y[1] == 1\}, y[x], x]$
 $\{\{y[x] \rightarrow (0/. \$Aborted[]) + (x - (0/. \$Aborted[])) \text{UnitStep}[x]\}\}$

или справляется

s = DSolve[{y'[x] + Max[x, 1]y[x] == 0, y[0] == 1}, y[x], x]

$$\left\{ \left\{ y[x] \rightarrow \begin{cases} e^{-x} & x \leq 1 \\ e^{-\frac{1}{2} - \frac{x^2}{2}} & \text{True} \end{cases} \right\} \right\}$$

Вместо имени константы интегрирования, выбираемой по-умолчанию, можно определить другой символ

DSolve[y'[x] == ay[x], y[x], x, GeneratedParameters → K]

DSolve[y'[x] == ay[x], y[x], x, DSolveConstants → K] (в старых версиях)

{{y[x] → e^{ax} K[1]}}

Вместо имени константы можно использовать функции, которым в качестве аргумента будут передаваться индексы (номера) постоянных. Удобно использовать функцию Subscript[a, n], которая возвращает запись вида a_n .

DSolve[y''[x] == y[x], y[x], x, GeneratedParameters → (Subscript[C, #]&)]

{{y[x] → e^x C₁ + e^{-x} C₂}}

В некоторых случаях *Mathematica* возвращает несколько решений:

DSolve[(∂_xy[x])² + (y[x])² == 1, y[x], x] (решаем уравнение $(y')^2 + y^2 = 1$)

{{y[x] → -Sin[x - C[1]], {y[x] → Sin[x + C[1]]}}

Заметим, что здесь потеряны два решения $y(x) = \pm 1$ (в Mathematica 9.0).

Чтобы набрать выражение $\partial_x y[x]$ вы можете набрать комбинацию *Esc* – *dt* – *Esc*. Она создает шаблон ввода ∂_{\square} , который вы должны заполнить $\partial_x y[x]$. Можно также ввести комбинацию *Esc* – *pd* – *Esc*, которая создает символ ∂ , затем комбинацию *Ctrl* – *_* (подчеркивание), затем *x* (появится ∂_x), затем *→* (клавиша управления текстовым курсором), затем введите $y[x]$.

Однако, привычнее использовать «'» (штрих) для обозначения производных функции в ОДУ.

DSolve[y[x]y'[x] == 1, y, x]

{{y → Function[{x}, -√2√x + C[1]], {y → Function[{x}, √2√x + C[1]]}}

DSolve может решать широкий спектр дифференциальных уравнений, и решение, обычно, получается в виде правила подстановок.

DSolve[y'[x] + y[x] == 1, y[x], x]

{{y[x] → 1 + e^{-x} C[1]}}

Здесь второй параметр функции DSolve был $y[x]$, а не просто y как в предыдущем примере. Допустимо использовать обе формы, однако между ними имеется различие. В последнем примере решение получается в виде правил для вычисления *выражения* $y[x]$ (не *функции* и не ее частных значений, таких как $y[0]$). Например:

y[x] + 2y'[x] + y[0]/.%

{1 + e^{-x} C[1] + y[0] + 2y'[x]}

Обратите внимание, что $y[0]$ и $y'[x]$ не были вычислены. Значки (символы) */.* обозначают подстановку в предшествующее выражение результата последнего вычисления (знак %).

Вы можете попросить DSolve определить *функцию* y (а не *выражение* $y[x]$) и она вернет правило для вычисления функции y .

DSolve[$y'[x] + y[x] == 1, y, x$]
 $\{\{y \rightarrow \text{Function}[\{x\}, 1 + e^{-x} C[1]]\}\}$

В некоторых старых версиях результат будет возвращен в виде

$\{\{y \rightarrow (1 + E^{-\#1} C[1]) \&\}\}$

Здесь символ #1 обозначает 1-й аргумент будущей функции, а знак амперсанда (&) постфиксное определение функции. Напомним, что формальные параметры при определении функции обозначаются # (или #1, #2 и т.д.), само имя функции обозначается #0, ## используется для обозначения всего списка аргументов, ##n обозначает списка аргументов начиная с n-того.

Последнее выражение можно прочесть так: везде вместо имени функции y следует подставлять выражение $(1 + e^{-\#1} C[1])$, которое определяет функцию и, где $C[1]$ символьная константа, #1 - имя первого (и единственного) аргумента функции. Полученное правило подстановки будет применяться везде, где имя функции y будет сочетаться с аргументом в квадратных скобках, например, $y[x]$ или $y[0]$.

Теперь в выражении

$y[x] + 2y'[x] + y[0] /. \%$

правило подстановки будет применяться и к начальному значению $y[0]$ и к производной $y'[x]$:

$\{2 + C[1] - e^{-x} C[1]\}$

Подстановка решения в уравнение в виде функции дает True.

$y'[x] + y[x] == 1 /. \%\%$
 $\{\text{True}\}$

Здесь символ %% (два процента) означает решение, полученное два шага назад. Итак,

DSolve[eqn, $y[x]$, x] решает ДУ относительно *выражения* $y[x]$.

DSolve[eqn, y , x] решает ДУ относительно *функции* y .

Приведем еще несколько примеров.

DSolve[$y''[x] + y[x] == 0, y[x], x, \text{GeneratedParameters} \rightarrow (\text{Subscript}[C, \#] \&)]$
 $\{\{y[x] \rightarrow \cos[x] C_1 + \sin[x] C_2\}\}$

DSolve[$\{\partial_{xx} y[x] + y[x] == 0, y[\pi/2] == 1, y'[\pi/2] == 0\}, y[x], x]$
 $\{\{y[x] \rightarrow \sin[x]\}\}$

Чтобы набрать выражение $\partial_{xx} y[x]$ вы можете ввести комбинацию *Esc* – *pd* – *Esc*, которая создает символ ∂ , затем комбинацию *Ctrl* – *_* (подчеркивание), затем x , затем комбинацию *Esc* – *занятая* – *Esc*, которая ставит невидимую запятую, затем снова x , после чего появится ∂_{xx} , затем \rightarrow (клавиша управления текстовым курсором), затем вводите $y[x]$.

DSolve может решать линейные обыкновенные дифференциальные уравнения с постоянными коэффициентами любого порядка. Функция может решать много линейных уравнений с непостоянными коэффициентами, а также может решать много нелинейных ОДУ, методы решения которых изложены в стандартных руководствах.

Вот несколько примеров. Общее решение уравнения четвертого порядка включает 4 неопределенные константы.

DSolve[$y''''[x] == y[x], y[x], x$]

{{ $y[x] \rightarrow e^x C[1] + e^{-x} C[3] + C[2] \cos[x] + C[4] \sin[x]$ }}

Каждое независимое начальное условие уменьшает на единицу количество независимых констант.

DSolve[$\{y''''[x] == y[x], y[0] == y'[0] == 0\}, y[x], x$]

{{ $y[x] \rightarrow e^{-x} (-C[1] + e^{2x} C[1] - C[2] + e^x C[2] \cos[x] - 2e^x C[1] \sin[x] - e^x C[2] \sin[x])$ }}

DSolve умеет решать краевые задачи

DSolve[$\{y''[x] + y[x] == 0, y[0] == 1, y[\pi/2] == 0\}, y[x], x$]

{{ $y[x] \rightarrow \cos[x]$ }}

Однако следует помнить, что не все краевые задачи имеют решение, например

DSolve[$\{y''[x] + y[x] == 0, y[0] == 1, y[\pi] == 0\}, y[x], x$]

DSolve::bvnul: For some branches of the general solution, the given boundary conditions lead to an empty solution

{}

или, наоборот, могут иметь не единственное решение

DSolve[$\{y''[x] + y[x] == 0, y[0] == 0, y[\pi] == 0\}, y[x], x$]

DSolve::bvsing: Unable to resolve some of the arbitrary constants in the general solution using the given boundary conditions...

{{ $y[x] \rightarrow C[2] \sin[x]$ }}

В граничных условиях можно задавать линейную комбинацию значений функции и ее производных.

DSolve[$\{y''[x] + y[x] == 0, y[0] == 1, y[\pi] + y'[\pi] == 0\}, y[x], x$]

{{ $y[x] \rightarrow \cos[x] - \sin[x]$ }}

Определение точного решения ОДУ является сложной задачей, и далеко не все уравнения имеют общее решение в символьном виде. Легче всего решаются линейные ОДУ

DSolve[$\{xy'[x] == 3y[x] + x^4 \cos[x], y[2\pi] == 0\}, y[x], x$]

{{ $y[x] \rightarrow x^3 \sin[x]$ }}

DSolve[$y'[x] - x y[x] == 0, y[x], x$]

{{ $y[x] \rightarrow e^{\frac{x^2}{2}} C[1]$ }}

Добавление неоднородности усложняет вид решения.

DSolve[$y'[x] - x y[x] == 1, y[x], x$]

{{ $y[x] \rightarrow e^{\frac{x^2}{2}} C[1] + e^{\frac{x^2}{2}} \sqrt{\frac{\pi}{2}} \operatorname{Erf}\left[\frac{x}{\sqrt{2}}\right]$ }}

Здесь $\operatorname{Erf}[x]$ – функция (интеграл) ошибок. Вот пример нелинейного ОДУ

DSolve[$x^2 y''[x] - y'[x]^2 == 0, y[x], x$]

{{ $y[x] \rightarrow -\frac{x}{C[1]} + C[2] - \frac{\operatorname{Log}[1 - x C[1]]}{C[1]^2}$ }}

Решение многих ДУ может быть представлено только через специальные функции. В следующем уравнении решение определяется через функции Эйри

DSolve[$y''[x] - x y[x] == 0, y[x], x$]

{{ $y[x] \rightarrow \operatorname{AiryAi}[x] C[1] + \operatorname{AiryBi}[x] C[2]$ }}

Решение следующего уравнения выражается через функции Бесселя.

DSolve[$x^2 y''[x] + x y'[x] + (x^2 - n^2) y[x] == 0, y[x], x$]

{{ $y[x] \rightarrow \operatorname{BesselJ}[n, x] C[1] + \operatorname{BesselY}[n, x] C[2]$ }}

Решения, возвращаемые функцией `DSolve`, иногда включает интегралы, которые не могут быть точно вычислены в символьном виде. Также иногда `DSolve` дает представление решения в неявной форме с использованием функции `Solve`. Следующее уравнение Абеля имеет представление решения в виде неявно определяемой функции.

`DSolve[y'[x] + x y[x]^3 + y[x]^2 == 0, y[x], x]`

`Solve[$\frac{1}{2} \left(\frac{2 \text{ArcTanh}\left[\frac{-1 - 2xy[x]}{\sqrt{5}}\right]}{\sqrt{5}} + \text{Log}\left[\frac{-1 - xy[x](-1 - xy[x])}{x^2 y[x]^2}\right] \right) == C[1] - \text{Log}[x], y[x]]$]`

Есть возможность создавать собственное обозначение для производных, которые можно использовать при записи ДУ.

`dx := D[#, x] &` (пользовательское обозначение функции дифференцирования)

`dx[f[x]]`

`f'[x]`

или

`dx = Function[{f}, D[f, x]]`

`dx@f[x]`

`f'[x]`

Здесь символ `@` обозначает префиксное использование/применение функции пользователя `dx` к последующему аргументу `f[x]`. В новых обозначениях например, запись

`dx@y[x] == -x^3 y[x]`

будет представлять дифференциальное уравнение

`y'[x] == -x^3 y[x]`

Допустимо использование полной формы функции дифференцирования, например

`Derivative[2][y][x]`

`y''[x]`

После решения ОДУ можно выполнить проверку полученного решения с помощью подстановки

`ds = DSolve[{y''[x] - y[x] == 0, y[0] == 1, y'[0] == 4}, y, x]`

`{{y -> Function[{x}, $\frac{1}{2} e^{-x} (-3 + 5 e^{2x})$]}}`

`{y''[x] - y[x] == 0, y[0] == 1, y'[0] == 4} /. ds`

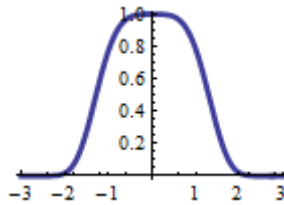
`{{True, True, True}}`

Если мы хотим построить график решения, то список правил подстановки надо преобразовать в выражение, которое следует передать функции `Plot`. Это делается следующим образом

`ds = DSolve[{y'[x] == -x^3 y[x], y[0] == 1}, y[x], x]`

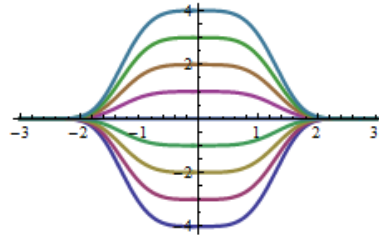
`{{y[x] -> $e^{-\frac{x^4}{4}}$ }}`

`Plot[y[x] /. ds, {x, -3, 3}, PlotStyle -> Thickness[0.01]]`



Можно построить график решений при различных значениях постоянной, вместо того, чтобы использовать параметр в начальных условиях.

```
ds = DSolve[{y'[x] == -x3y[x]}, y[x], x];
tab = Table[y[x]/.ds[[1]]/.{C[1] → k}, {k, -4, 4, 1}]
Plot[Evaluate[tab], {x, -3, 3}, PlotStyle → {Thickness[0.01]}]
{-4e-x4/4, -3e-x4/4, -2e-x4/4, -e-x4/4, 0, e-x4/4, 2e-x4/4, 3e-x4/4, 4e-x4/4}
```

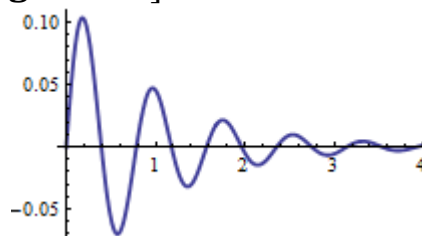


Часто нам требуется решение в виде функции или выражения, которые мы собираемся использовать не только для построения графика. Например, при построении фазовых кривых ДУ 2 – го порядка нам нужна еще и производная решения. Тогда список правил подстановки можно преобразовать в функцию.

```
ds = DSolve[{y''[x] + 2y'[x] + 65y[x] == 0, y[0] == 0, y'[0] == 1}, y, x]
{{y → Function[{x},  $\frac{1}{8}e^{-x}\text{Sin}[8x]$ ]}}
```

Для того, чтобы последний список списков преобразовать в одинарный список надо избавиться от одной пары охватывающих фигурных скобок, например, взять `ds[[1]]`. Можно избавиться от обеих пар охватывающих скобок, взяв `ds[[1, 1]]`. Подставив этот результат вместо `y` с помощью операции подстановки `/.` (слэш и точка), получим функцию, представляющую решение ОДУ

```
z = y/.ds[[1, 1]]
Plot[z[x], {x, 0, 4}, PlotRange → All]
```



Созданную функцию `z` вы можете использовать для дифференцирования, интегрирования, вычисления значения в точке и в любых других ситуациях, в которых уместно использовать функцию.

Ту же функцию можно создать следующим образом

```
z[x_] := (y/.ds[[1]])[x]
```

Мы создали функцию `z[x]`, представляющую решение, используя подстановку `{y→Function[...]}`.

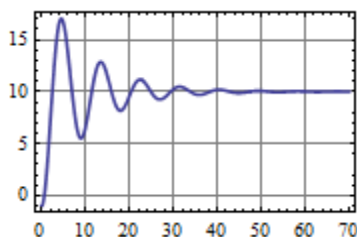
Решение можно искать в виде выражения $y[x]$.

```
ds = DSolve[{4y''[x] + 4/5y'[x] + 2y[x] == 20, y[0] == -1, y'[0] == 0}, y[x], x]
{{y[x] -> 1/7 e^{-x/10} (70 e^{x/10} - 77 Cos[7x/10] - 11 Sin[7x/10])}}
```

Тогда для построения функции, представляющей решение, можно поступить следующим образом

```
z[t_]:= (y[x]/. ds[[1]])/. x -> t
```

```
Plot[z[x], {x, 0, 70}, PlotRange -> All, PlotStyle -> Thickness[0.01],
GridLines -> Automatic, Frame -> True]
```



Не забывайте удалять значения переменных из рабочего пространства. Так перед предыдущим кодом следует выполнить команду `Remove[z]`, чтобы «забыть» определение функции `z`, сделанное в предыдущем примере.

Функцию решения можно создать еще так

```
z[x_] = y[x]/. ds[[1, 1]]
```

```
Plot[z[x], {x, 0, 70}, PlotRange -> All]
```

Построить график можно и следующими командами

```
z[x_] := y[x]/. ds[[1]] (* отложенное присваивание *)
```

```
Plot[Evaluate[z[x]], {x, 0, 70}, PlotRange -> All]
```

Однако «настоящей» функции команда `z[x_] := y[x]/. ds[[1, 1]]` не создает, т.к. значение функции в точке, например, `z[u]`, или производная функции `z'[x]` вычисляться не будут. При построении графика команда `Evaluate[z[x]]` заставляет ядро *Mathematica* сначала вычислить `z[x]` (создать и загрузить выражение `z[x]` в рабочее пространство) и только потом функция `Plot` использует ее для построения графика.

Есть еще один способ построения функции, представляющей решение ДУ. Для этого надо выделить *выражение* из правой части подстановки `y[x] -> выражение`, которое затем использовать при создании функции

```
ds = DSolve[{y''[x] + 2x y'[x] == 0, y[0] == 0, y'[0] == 1}, y[x], x]
```

```
{{y[x] -> 1/2 sqrt(pi) Erf[x]}}
```

Полная форма последнего результата может быть получена следующей командой

```
FullForm[ds[[1, 1]]]
```

```
Rule[y[x], Times[Rational[1,2], Power[Pi, Rational[1,2]], Erf[x]]]
```

Замена имени функции «Rule» на имя функции «List» вернет результат в виде списка, составленного из левой и правой частей подстановки

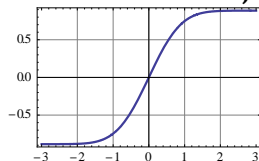
```
{l, r} = ds[[1, 1]]/. {Rule -> List}
```

```
{y[x], 1/2 sqrt(pi) Erf[x]}
```

Второй элемент **r** списка содержит формульное выражение решения и его можно использовать для построения функции

z[t_]:=r/.x→t

Plot[z[x], {x, -3, 3}, GridLines->Automatic, Frame->True]



Тот же график можно построить следующим образом

Plot[r, {x, -3, 3}, GridLines->Automatic, Frame->True]

Кроме графиков решений, полезны графики фазовых траекторий. Для ДУ 2-го порядка они строятся на плоскости (y, y') . Вот как можно построить фазовую траекторию, используя созданную функцию решения.

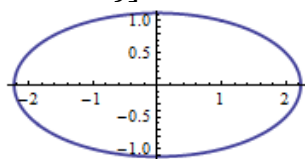
ds = DSolve[{y''[x] + $\frac{1}{4}$ y[x] == 0, y[0] == 1, y'[0] == 1}, y, x]

{{y → Function[{x}, Cos[$\frac{x}{2}$] + 2Sin[$\frac{x}{2}$]]}}

X[t_]:= (y/.ds[[1]])[t]

Y[t_]:= X'[t]

ParametricPlot[{X[t], Y[t]}, {t, 0, 4π}]



Покажем еще один способ выделения правой части подстановки вида $y[x] \rightarrow \text{выражение}$. Для этого вспомним, что к элементам выражений мы можем обращаться как к элементам списков. Например

z = f[g[a], h[b]]

{z[[0]], z[[1]], z[[2]], z[[2, 0]], z[[2, 1]]}

{f, g[a], h[b], h, b}

Здесь для выделения элементов выражения **z** мы используем индексацию. Так **z[[0]]**=f – имя внешней функции; **z[[1]]**=g[a] – первый аргумент функции, **z[[2]]**=h[b] – второй аргумент; **z[[2, 0]]**=h – имя функции, стоящей во втором аргументе; **z[[2, 1]]**=b – аргумент второй функции. Поскольку запись $y[x] \rightarrow \text{выражение}$ на самом деле имеет форму

Rule[y[x], выражение] (внешняя функция **Rule**), то для выделения правой части подстановки можно использовать подходящую индексацию. Например,

ds = DSolve[{y'[x] == a y[x], y[c] == b}, y[x], x]

{{y[x] → $b e^{-ac+ax}$ }}

Тогда

ds[[1, 1]]

y[x] → $b e^{-ac+ax}$

и

ds[[1, 1]][[1]]

y[x]

Наконец выделяем правую часть из подстановки

```
ds[[1,1]][[2]]
```

```
b $e^{-ac+ax}$ 
```

Решим ДУ и построим его фазовую траекторию

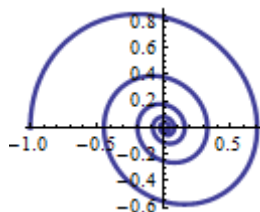
```
ds = DSolve[{64y''[x] + 16y'[x] + 65y[x] == 1, y[0] == -1, y'[0] == 0}, y[x], x]
```

```
{{y[x] →  $\frac{1}{260} e^{-x/8} (4e^{x/8} - 264\text{Cos}[x] - 33\text{Sin}[x])$ }}
```

```
X[x_] = ds[[1,1]][[2]]
```

```
Y[x_] := X'[x]
```

```
ParametricPlot[{X[t], Y[t]}, {t, 0, 30}, PlotStyle -> Thickness[0.01]]
```



Для выделения выражения, стоящего в правой части подстановки `ds`, мы использовали индексацию `ds[[1,1]][[2]]`. Конечно допустимо использовать `ds[[1,1,2]]`. Обратите также внимание на присваивание при создании функции `X[x]`. Если при создании функции использовать отложенное присваивание «`:=`», то код работать не будет, т.к. функция должна быть вычислена и загружена в рабочее пространство до дифференцирования в следующей строке.

Если решение получено в неявном виде, то графическая интерпретация тоже может быть полезна.

```
s = DSolve[y'[x] ==  $\frac{x^2 - 2y[x]^2 + 14}{\text{Sin}[y[x]] + 4xy[x] - 1}$ , y[x], x]
```

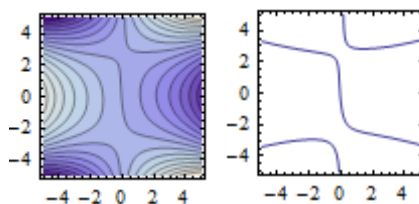
```
Solve[- $\frac{x^3}{3}$  - Cos[y[x]] - y[x] + 2x(-7 + y[x]^2) == C[1], y[x]]
```

Решение ОДУ было сведено к решению алгебраического уравнения. Можно построить контурный график левой части алгебраического уравнения или только «нулевые» линии контура

```
sn = s[[1,1]]/. {y[x] → y}
```

```
ContourPlot[sn, {x, -5, 5}, {y, -5, 5}, Contours -> 15]
```

```
ContourPlot[sn == 0, {x, -5, 5}, {y, -5, 5}]
```



Иногда решение ОДУ возвращается с использованием объекта `InverseFunction` (обратная функция).

```
sol = DSolve[y''[x] + y[x]y'[x]^4 == 0, y, x]
{{y -> Function[{x}, InverseFunction[C[1]Log[#1 + Sqrt[-2C[1] + #1^2]
- 1/2 #1 Sqrt[-2C[1] + #1^2]&][x + C[2]]]},
{ y -> Function[{x}, InverseFunction[-C[1]Log[#1 + Sqrt[-2C[1] + #1^2]
+ 1/2 #1 Sqrt[-2C[1] + #1^2]&][x + C[2]]]}}
```

Чтобы записать его в более привычном виде, выполним команды

```
soly = sol[[1, 1, 2, 2]]
```

```
InverseFunction[C[1]Log[#1 + Sqrt[-2C[1] + #1^2] - 1/2 #1 Sqrt[-2C[1] + #1^2]&][x + C[2]]
```

Чтобы понять, что делает последняя команда, мы не стали завершать ее точкой с запятой. Она выделила выражение обратной функции (присутствует аргумент функции) из правила подстановки $y \rightarrow \dots$. Для понимания происходящего выполним команду

```
Head[soly]
```

```
InverseFunction[C[1]Log[#1 + Sqrt[-2C[1] + #1^2] - 1/2 #1 Sqrt[-2C[1] + #1^2]&
```

Эта команда выделяет «чистую» обратную функцию (нет именованного аргумента). Наконец команда

```
Head[soly][[1]][y[x]] == soly[[1]]
```

```
C[1]Log[y[x] + Sqrt[-2C[1] + y[x]^2] - 1/2 y[x] Sqrt[-2C[1] + y[x]^2] == x + C[2]
```

выделяет выражение неявного уравнения. Фактически, целью преобразований было выделение аргумента функции InverseFunction командой Head[soly][[1]].

Последнее выражение представляет неявное выражение функции $y(x)$ через x и две произвольные постоянные. Полученное уравнение можно представить графически с использованием контурного графика (при некоторых значениях постоянных).

```
eq = Head[soly][[1]][y] == soly[[1]] /. {C[1] -> 1, C[2] -> 1};
```

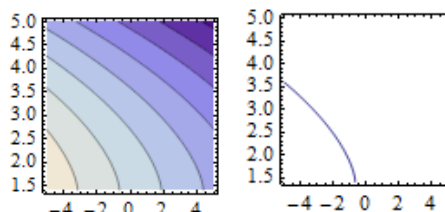
```
z = eq[[1]] - eq[[2]]
```

```
-1 - x - 1/2 y Sqrt[-2 + y^2] + Log[y + Sqrt[-2 + y^2]]
```

В последней строке записана левая часть неявного уравнения. Теперь для этого выражения можно строить контурные графики.

```
ContourPlot[z, {x, -5, 5}, {y, Sqrt[2], 5}]
```

```
ContourPlot[z == 0, {x, -5, 5}, {y, Sqrt[2], 5}]
```



Правый график представляет график решения.

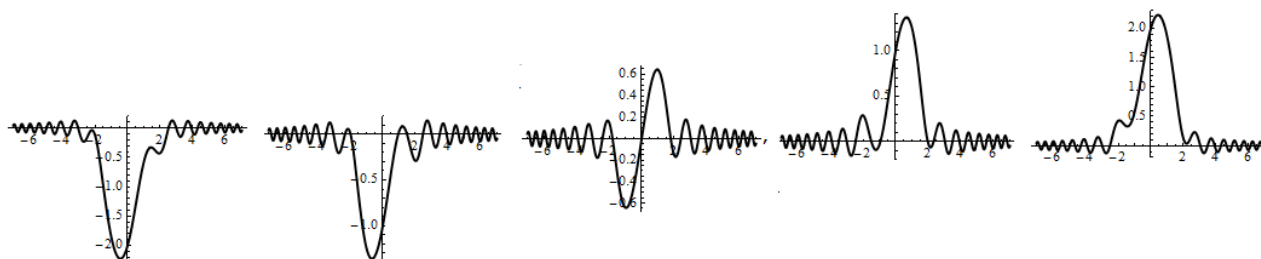
Для построения нескольких графиков можно использовать функцию Table. Она создает списки, элементами которых могут быть объекты различной природы, в частности это могут быть графические объекты (графики).

```
sol = DSolve[y'[x] + xy[x] == Cos[x^2], y, x]
```

```
ts = Table[sol/. C[1] → i, {i, -4, 4, 2}];
```

```
Table[Plot[Evaluate[y[x]/. ts[[i]]], {x, -7, 7}], {i, 5}]
```

$$\{ \{ y \rightarrow \text{Function}[\{x\}, e^{-\frac{x^2}{2}} C[1] + \frac{1}{2} e^{-\frac{x^2}{2}} \sqrt{\frac{\pi}{10}} (\sqrt{1 + 2i} \text{Erfi}[\sqrt{\frac{1}{2} - ix}] + \sqrt{1 - 2i} \text{Erfi}[\sqrt{\frac{1}{2} + ix}])] \} \}$$



Обратите внимание, что в выражении решения присутствует мнимая единица таким образом, что решение все равно остается вещественным.

Другие функции, связанные с поиском решений ОДУ.

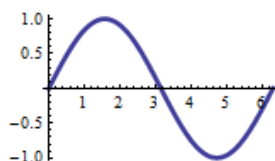
Функция DifferentialRoot представляет функцию, которая является решением линейного ОДУ.

Вот пример ДУ, решением которого является Sin[x].

```
f = DifferentialRoot[Function[{y, x}, {y''[x] + y[x] == 0, y[0] == 0, y'[0] == 1}]]
```

```
DifferentialRoot[Function[{y, x}, {y''[x] + y[x] == 0, y[0] == 0, y'[0] == 1}]]
```

```
Plot[f[x], {x, 0, 2Pi}]
```



DifferentialRoot – работает как функция

```
f[{ $\frac{\pi}{6}$ ,  $\frac{\pi}{4}$ ,  $\frac{\pi}{3}$ ,  $\frac{\pi}{2}$ }]
```

```
{ $\frac{1}{2}$ ,  $\frac{1}{\sqrt{2}}$ ,  $\frac{\sqrt{3}}{2}$ , 1}
```

```
DifferentialRoot[Function[{y, x}, {y''[x] + y[x] == 0, y[0] == 0, y'[0] == 1}]] [1]
```

```
Sin[1]
```

```
f[x]//FunctionExpand
```

```
Sin[x]
```

Функция FunctionExpand, использованная здесь, умеет упрощать выражения, содержащие специальные, тригонометрические и некоторые другие функции (такие как DifferentialRoot).

Функция `DifferentialRootReduce[expr, x]` пытается привести выражение `expr` к виду `DifferentialRoot` как функции `x`. Используя функцию `DifferentialRootReduce` можно получить задачу (дифференциальное уравнение и начальные условия), которому удовлетворяет функция

`DifferentialRootReduce[Cos[x], x][[0, 1]]`

`DifferentialRoot[Function[{y, x}, {y[x] + y''[x] == 0, y[0] == 1, y'[0] == 0}]] [x]`

`%[[0, 1]][y, x]`

`{y[x] + y''[x] == 0, y[0] == 1, y'[0] == 0}`

Чтобы понять, как работает последняя введенная команда мы должны вспомнить как работает индексация в выражениях. Команда `%[[0]]` вернет функцию `DifferentialRoot[Function[{y, x}, ...]]`, а команда `%[[0, 1]]` вернет функцию `Function[{y, x}, ...]`. Использование команды `%[[0, 1]][y, x]` возвращает выражение, вычисляемое функцией, т.е. `{y[x] + y''[x] == 0, y[0] == 1, y'[0] == 0}`.

Вот еще примеры на использование этой функции для получения «краевой задачи»

`DifferentialRootReduce[Exp[-x]Cos[x], x][[0, 1]][y, x]`

`{2y[x] + 2y'[x] + y''[x] == 0, y[0] == 1, y'[0] == -1}`

`DifferentialRootReduce[Sqrt[x], x][[0, 1]][y, x]`

`{-y[x] + 2xy'[x] == 0, y[1] == 1}`

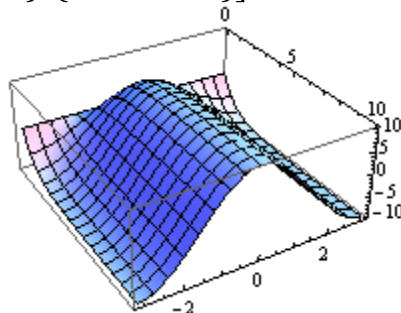
Искомая функция может зависеть от параметра. Например

`sol = DSolve[{D[x[t, α], {t, 2}] == -D[x[t, α], t],`

`x[0, α] == 0, Derivative[1, 0][x][0, α] == 10 Cos[α]}, x[t, α], t]`

`{{x[t, α] → 10e-t(-1 + et)Cos[α]}}`

`Plot3D[x[t, α]/.sol, {t, 0, 10}, {α, -Pi, Pi}]`



Следует помнить, что в общем виде решения уравнения с параметром могут обнаружиться значения параметра, для которых решение недопустимо. Например

`sol = DSolve[{y'[x] == xy[x], y[0] == 1/a}, y, x]`

`{{y → Function[{x}, $\frac{e^{\frac{x^2}{2}}}{a}$]}}`

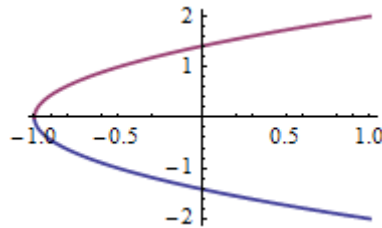
Очевидно, что значение $a=0$ недопустимо. В этом примере это видно из постановки задачи, однако могут встретиться и более сложные случаи.

Приведем еще несколько замечаний, касающихся функции DSolve.

1⁰. Даже для простых уравнений функция DSolve может найти несколько решений.

```
gs = DSolve[{y'[x] == 1/y[x]}, y, x]
{{y → Function[{x}, -√2√x + C[1]], {y → Function[{x}, √2√x + C[1]]}}
```

```
Plot[Evaluate[y[x]/.gs/.{C[1] → 1}], {x, -1, 1}]
```



Задание начального условия выделяет одну ветвь решения

```
gs = DSolve[{y'[x] == 1/y[x], y[0] == 1}, y, x]
DSolve::bvnul: For some branches of the general solution,
the given boundary conditions lead to an empty solution.
```

```
{{y → Function[{x}, √1 + 2x]}}
```

При этом мы получаем сообщение, что могут существовать другие ветви решения (возможно решение не единственно). Если вы не желаете видеть сообщение, то используйте функцию Quiet.

```
gs = DSolve[{y'[x] == 1/y[x], y[0] == 1}, y, x]//Quiet
{{y → Function[{x}, √1 + 2x]}}
```

2⁰. В системе уравнений вместо запятой можно использовать символы && (логическое И). Так следующие две команды дают одинаковый результат.

```
DSolve[{y'[x] == y[x] && y[0] == 1}, y, x]
DSolve[{y'[x] == y[x], y[0] == 1}, y, x]
{{y → Function[{x}, ex]}}
```

3⁰. Иногда, когда функция DSolve не дает результата, можно попробовать решить задачу относительно обратной функции, поменяв ролями зависимую и независимую переменные.

```
DSolve[y'[x] == 1/(x - y[x]) && y[0] == 1, y, x]
Solve::ifun: Inverse functions are being used ...
{ }
DSolve[x'[y] == (x[y] - y) && x[1] == 0, x, y]
{{x → Function[{y}, (e - 2ey + ey)/e]}}
```

Во втором случае мы записали задачу относительно функции $x(y)$.

4.1.2 Символьное решение систем ОДУ

Функция `DSolve[{eqn1, eqn2, ... }, {y1, y2, ...}, x]` умеет решать системы дифференциальных уравнений, которые при записи необходимо объединить в список. В таком случае в качестве второго аргумента `Dsolve` надо задавать список имен искомых функций.

DSolve[$\{y'[x] == -z[x], z'[x] == -y[x]\}, \{y[x], z[x]\}, x]$

$\{y[x] \rightarrow \frac{1}{2}e^{-x}(1 + e^{2x})C[1] - \frac{1}{2}e^{-x}(-1 + e^{2x})C[2],$
 $z[x] \rightarrow -\frac{1}{2}e^{-x}(-1 + e^{2x})C[1] + \frac{1}{2}e^{-x}(1 + e^{2x})C[2]\}$

Здесь в списке правил подстановок используются выражения. Но решение той же системы можно получить с использованием функций

DSolve[$\{y'[x] == -z[x], z'[x] == -y[x]\}, \{y, z\}, x]$

$\{y \rightarrow \text{Function}[\{x\}, \frac{1}{2}e^{-x}(1 + e^{2x})C[1] - \frac{1}{2}e^{-x}(-1 + e^{2x})C[2]],$
 $z \rightarrow \text{Function}[\{x\}, -\frac{1}{2}e^{-x}(-1 + e^{2x})C[1] + \frac{1}{2}e^{-x}(1 + e^{2x})C[2]]\}$

Заметим, что в обоих случаях решение включает две константы. Та же система с начальными условиями даст решение без произвольных постоянных

DSolve[$\{y'[x] == -z[x], z'[x] == -y[x], y[0] == 0, z[0] == 1\}, \{y[x], z[x]\}, x]$

$\{y[x] \rightarrow -\frac{1}{2}e^{-x}(-1 + e^{2x}), z[x] \rightarrow \frac{1}{2}e^{-x}(1 + e^{2x})\}$

Для того, чтобы полученные правила подстановки преобразовать в функции надо избавиться от внешнего списка, а из внутреннего взять первое или второе правило для подстановки соответственно в функции x и y . Например

ds = DSolve[$\{x'[t] == y[t], y'[t] == x[t], x[0] == 0, y[0] == 1\}, \{x, y\}, t]$

$\{x \rightarrow \text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + e^{2t})], y \rightarrow \text{Function}[\{t\}, \frac{1}{2}e^{-t}(1 + e^{2t})]\}$

Выражение для представления решения мы можем получить, используя индексацию, например, так `ds[[1, 1, 2]][t]`. Следующая цепочка команд и их результатов поясняет это.

ds[[1, 1]]

$x \rightarrow \text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + e^{2t})]$

ds[[1, 1, 2]]

$\text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + e^{2t})]$

ds[[1, 1, 2]][t]

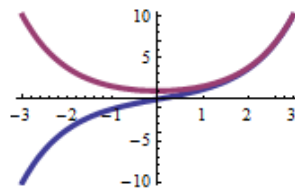
$\frac{1}{2}e^{-t}(-1 + e^{2t})$

Если мы захотим построить графики функций $x[t]$, $y[t]$, представляющих решение системы ОДУ, то эти функции надо определить

x := **ds**[[1, 1, 2]]

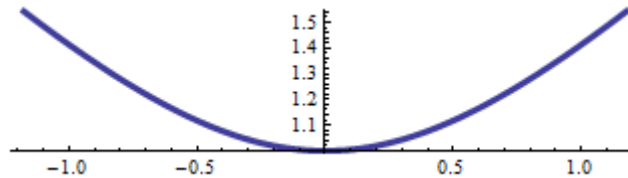
y := **ds**[[1, 2, 2]]

Plot[$\{x[t], y[t]\}, \{t, -3, 3\}, \text{PlotStyle} \rightarrow \text{Thickness}[0.01]$]



Фазовая траектория, соответствующая решению системы, может быть построена с помощью функции `ParametricPlot[...]`.

ParametricPlot[$\{x[t], y[t]\}, \{t, -1, 1\}, \text{PlotStyle} \rightarrow \text{Thickness}[0.01]$]



Построить фазовую траекторию можно без создания функций, представляющих решение

ds = DSolve[$\{x'[t] == y[t], y'[t] == x[t], x[0] == 0, y[0] == 1\}, \{x, y\}, t]$
 $\{\{x \rightarrow \text{Function}[\{t\}, \frac{1}{2} e^{-t}(-1 + e^{2t})], y \rightarrow \text{Function}[\{t\}, \frac{1}{2} e^{-t}(1 + e^{2t})]\}$

ParametricPlot[$\{x[t], y[t]\} /. \text{ds}, \{t, -1, 1\},$
PlotStyle $\rightarrow \text{Thickness}[0.01], \text{PlotRange} \rightarrow \text{All}]$

Вот еще один пример решения системы двух ДУ первого порядка

Remove[$x, y, t]$

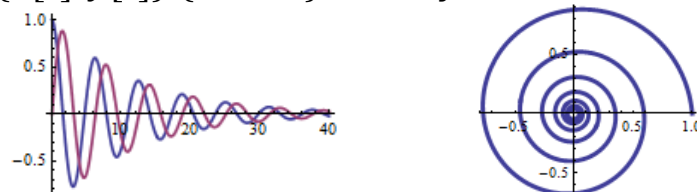
n := 12

ds = DSolve[$\{nx'[t] + x[t] + ny[t] == 0, ny'[t] - nx[t] + y[t] == 0,$
 $x[0] == 1, y[0] == 0\}, \{x, y\}, t]$

$x = \text{ds}[[1, 1, 2]]$; $y = \text{ds}[[1, 2, 2]]$

Plot[$\{x[t], y[t]\}, \{t, 0, 40\}, \text{PlotStyle} \rightarrow \text{Thickness}[0.01], \text{PlotRange} \rightarrow \text{All}]$

ParametricPlot[$\{x[t], y[t]\}, \{t, 0, 40\}, \text{PlotStyle} \rightarrow \text{Thickness}[0.02]$]



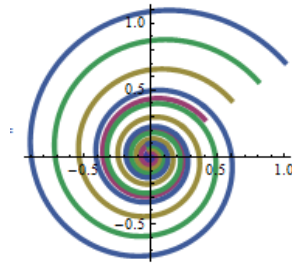
Вот как можно построить сразу несколько фазовых траекторий, соответствующих разным начальным значениям

Remove[$x, y, t]$

n := 8

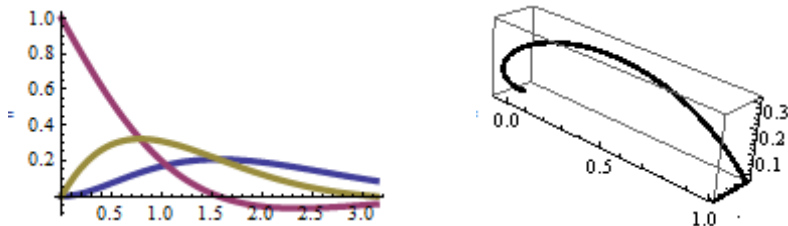
ds = DSolve[$\{nx'[t] + x[t] + ny[t] == 0, ny'[t] - nx[t] + y[t] == 0,$
 $x[0] == a, y[0] == a \cos[\frac{\pi}{4}]\}, \{x, y\}, t]$

ParametricPlot[**Table**[$\{x[t], y[t]\} /. \text{ds} /. \{a \rightarrow m\},$
 $\{m, 0.2, 1, 0.2\}], \{t, 0, 20\}, \text{Evaluated} \rightarrow \text{True}, \text{PlotRange} \rightarrow \text{All}]$



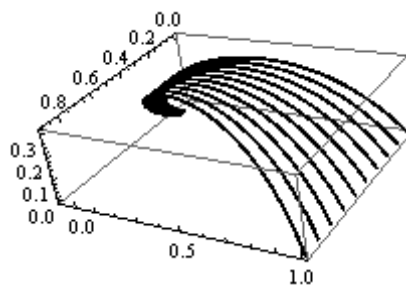
Вот пример построения фазовой траектории для системы 3 – х уравнений первого порядка.

```
Remove[x, y, z, t]
ds = DSolve[{x'[t] == -x[t] + z[t], y'[t] == -y[t] - z[t],
            z'[t] == y[t] - z[t], x[0] == 0, y[0] == 1, z[0] == 0}, {x, y, z}, t]
{{x → Function[{t}, -e-t(-1 + Cos[t])], y → Function[{t}, e-tCos[t]],
  z → Function[{t}, e-tSin[t]]}}
x = ds[[1, 1, 2]]
y = ds[[1, 2, 2]]
z = ds[[1, 3, 2]]
Plot[{x[t], y[t], z[t]}, {t, 0, π}, PlotStyle → Thickness[0.02], PlotRange → All]
ParametricPlot3D[{x[t], y[t], z[t]}, {t, 0, π}, PlotStyle → Thickness[0.02]]
```



В следующем примере мы строим несколько фазовых траекторий той же системы ДУ, соответствующих разным начальным условиям.

```
Remove[x, y, z, fx, fy, fz, t]
sys := {x'[t] == -x[t] + z[t], y'[t] == -y[t] - z[t], z'[t] == y[t] - z[t]}
col := 10
bc := Table[{x[0] == 0.1(i - 1), y[0] == 1, z[0] == 0}, {i, 1, col}]
pp = Table[se = Join[sys, bc[[i]]];
           u = DSolve[se, {x, y, z}, t];
           fx = u[[1, 1, 2]]; fy = u[[1, 2, 2]]; fz = u[[1, 3, 2]];
           p = ParametricPlot3D[{fx[t], fy[t], fz[t]}, {t, 0, π};
           p, {i, 1, col}];
Show[pp, PlotRange → All, AspectRatio → Automatic]
```



Систему ОДУ можно записывать, используя матрично – векторные обозначения

```

A = {{2, -3}, {1, -2}}
X[t_] = {x[t], y[t]};
sys = MapThread[#1 == #2 &, {X'[t], A.X[t]}]
{x'[t] == 2x[t] - 3y[t], y'[t] == x[t] - 2y[t]}
Теперь систему можно решить
sol = DSolve[sys, {x, y}, t]
{{x -> Function[{t},  $\frac{1}{2}e^{-t}(-1 + 3e^{2t})C[1] - \frac{3}{2}e^{-t}(-1 + e^{2t})C[2]$ ],
  y -> Function[{t},  $\frac{1}{2}e^{-t}(-1 + e^{2t})C[1] - \frac{1}{2}e^{-t}(-3 + e^{2t})C[2]$ ]}}

```

4.2 Численные решения

4.2.1 Численное решение ОДУ

В тех случаях, когда символьное решение дифференциального уравнения не может быть найдено помогает функция `NDSolve`, которая позволяет численно решать дифференциальные уравнения. Она имеет следующий синтаксис

```
NDSolve[{уравнения}, y, {x, x_min, x_max}]
```

где уравнения содержит запись ОДУ и начальных/граничных условий относительно неизвестной функции y и независимой переменной x , изменяющейся в пределе от x_{\min} до x_{\max} . `NDSolve` дает решение в виде объекта/функции `InterpolatingFunction`. Список уравнений должен содержать достаточное количество начальных и/или граничных условий для полного определения решения. Начальные и граничные условия обычно задаются в виде $y[x_0] == c_0$, $y'[x_0] == b_0$ и т.д.

Функция `NDSolve` имеет такие же аргументы, как и `DSolve`, но с некоторыми ограничениями. Надо определить достаточно начальных/граничных условий, чтобы получить полностью определенное решение. Надо также определить интервал изменения независимой переменной (на этом интервале решение должно существовать).

```

nds = NDSolve[{y'[x] == x^2, y[0] == 1}, y[x], {x, -3, 3}]
{{y[x] -> InterpolatingFunction[{{-3., 3.}}, " <> "][x]}}

```

Результат выдается в таком же формате, что и для `Solve` или `DSolve`, за исключением того, что решение является объектом типа `InterpolatingFunction`. Это черный ящик: задаешь входное значение и получаешь выходное, но не можешь видеть ничего внутри. Обычно лучшим способом использования `InterpolatingFunction` считается построение графика или вычисление значений функции в точках. Например, создадим функцию

```

z[x_] = nds[[1, 1, 2]]
{z[0], z[1], z[3]}
{1., 1.33333, 10.}

```

Начнем рассмотрение примеров.

```
nds = NDSolve[{y'[x] == y[x], y[0] == 2}, y, {x, 0, 3}]
{{y → InterpolatingFunction[{{0., 3.}}, " <> " ]}}
```

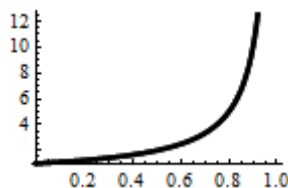
Решение представлено в виде списка подстановок, основным элементом которого является объект `InterpolatingFunction`. Для превращения списка подстановок в функцию надо поступить обычным способом

```
z = nds[[1, 1, 2]]
{z[0], z[1], z[3]}
{2., 5.43656, 40.17107}
```

Сравните вид искомого решения и способ создания функции `z` в этом и предыдущем примерах.

Сравним решения, полученные с помощью функции `Dsolve` и `NDSolve`.

```
nds = NDSolve[{y'[x] == y[x]^2, y[0] == 1}, y, {x, 0, 1}]
ds = DSolve[{y'[x] == y[x]^2, y[0] == 1}, y, x]
zn = nds[[1, 1, 2]]
zd = ds[[1, 1, 2]]
Plot[{zn[x], zd[x]}, {x, 0, 1}, PlotStyle → {{Black, Thickness[0.02]}]}
{{y → InterpolatingFunction[{{0., 1.}}, <> ]}}
{{y → Function[{x},  $\frac{1}{1-x}$ ]}}
```



Графики обеих функций сливаются. Однако, обратите внимание на пределы изменения параметра `x` при численном решении. Если определить верхнюю границу изменения `x` равной 1, то будем получать сообщения о возможных ошибках, которые в данном случае означают, что точка `x=1` является особой.

Дополнительные условия можно задавать в разных точках

```
nds = NDSolve[{y'''[x] + Sin[x] == 0, y[0] == 4, y[1] == 7, y[2] == 0},
y, {x, 0, 2}]
```

```
{{y → InterpolatingFunction[{{0., 2.}}, " <> " ]}}
```

В граничных условиях можно использовать линейные комбинации значений функции и ее производных.

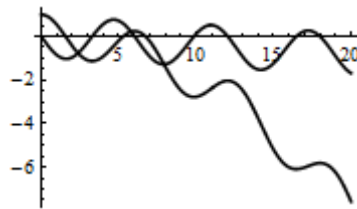
```
nds = NDSolve[{y''[x] + y[x] == 12x, 2y[0] - y'[0] == -1,
2y[1] + y'[1] == 9}, y, {x, 0, 1}]
```

```
{{y → InterpolatingFunction[{{0., 1.}}, " <> " ]}}
```

Если все начальные условия заданы в одной точке, скажем в точке `x0`, тогда для задания диапазона изменения независимой переменной можно использовать только одно значение, например, в виде `{x, x1}`. *Mathematica* сгенерирует решение в диапазоне `{x, x0, x1}`.

```
nds = NDSolve[{y''[t] + y[t] == -0.02t^2, y[0] == 1, y'[0] == 0}, y, {t, 20}];
```

```
Plot[{y[t], y'[t]}/. nds, {t, 0, 20}, PlotStyle → {Thickness[0.01], RGBColor[0, 0, 0]}]
```



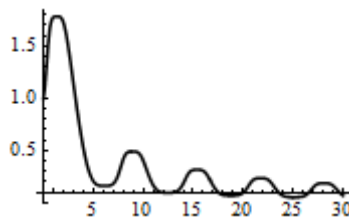
Попутно обратите внимание, что объект `InterpolatingFunction` можно дифференцировать.

Чтобы во всех примерах при построении графиков решений и фазовых траекторий постоянно не устанавливать длинный список опций, их можно установить один раз на всю рабочую сессию функцией `SetOptions`, например, так

```
SetOptions[Plot, PlotRange → All, PlotStyle → {Thickness[0.01], RGBColor[0, 0, 0]}];
```

Первый аргумент `SetOptions` – имя функции для которой будут изменены опции по-умолчанию, затем следует последовательность изменяемых опций и их значений. Теперь при построении графиков функцией `Plot` мы будем строить кривые черным цветом и с увеличенной толщиной.

```
s = NDSolve[{y'[x] == Sin[x + y[x]]^3 y[x], y[0] == 1}, y, {x, 0, 30}]  
Plot[Evaluate[y[x]/.s], {x, 0, 30}]
```

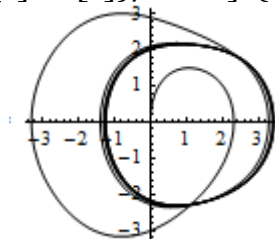


Используя функцию `SetOptions`, установим опции для функции `ParametricPlot`.

```
SetOptions[ParametricPlot, PlotRange → All, PlotStyle → {Thickness[0.01], Black}];
```

При построения фазовой траектории можно непосредственно дифференцировать объект `InterpolatingFunction`.

```
nds = NDSolve[{x''[t] + x'[t] + Cos[x[t]] == 4Cos[t],  
                  x[0] == x'[0] == 0}, x, {t, 0, 50}];  
ParametricPlot[Evaluate[{x[t], x'[t]}/.nds], {t, 0, 50}]
```



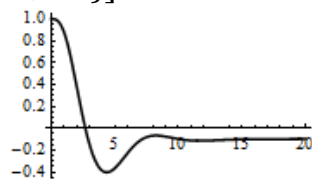
Вот пример уравнения, которое не удастся решить в символьном виде

```
DSolve[{y'''[x] + y''[x] + y'[x] == -y[x]^3, y[0] == 1, y'[0] == y''[0] == 0}, y, x]  
DSolve[{y'[x] + y''[x] + y^(3)[x] == -y[x]^3, y[0] == 1, y'[0] == y''[0] == 0}, y, x]
```

Когда *Mathematica* не может решить уравнение она возвращает исходный текст.

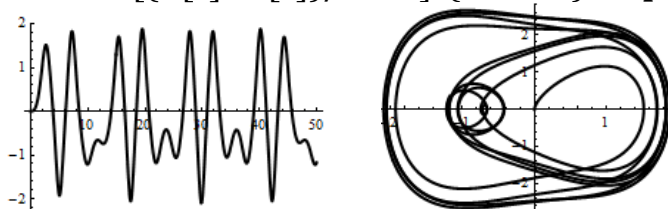
Численное решение этого уравнения у *Mathematica* не вызывает затруднений. Однако такое решение не представляет интереса, если его не использовать в привычном для нас виде, например в виде графика.

```
nds = NDSolve[{y'''[x] + y''[x] + y'[x] == -y[x]^3, y[0] == 1,
               y'[0] == y''[0] == 0}, y, {x, 0, 20}]
{{y → InterpolatingFunction[{{0., 20.}}, " <> " ]}}
Plot[Evaluate[y[x]/. nds], {x, 0, 20}]
```



Кроме графиков решений интерес представляют фазовые траектории, которые для ДУ 2-го порядка строятся на плоскости (x, x') с использованием функции ParametricPlot.

```
nds = NDSolve[{x''[t] + x[t]^3 == Sin[t], x[0] == x'[0] == 0}, x, {t, 0, 50}]
Plot[x[t]/. nds, {t, 0, 50}]
ParametricPlot[Evaluate[{x[t], x'[t]}/. nds], {t, 0, 50}, AspectRatio → 0.7]
```

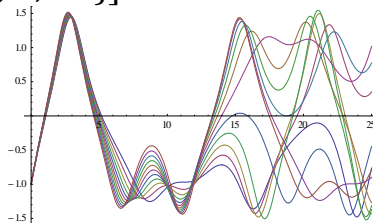


Здесь слева показан график решения, а справа – фазовая траектория. Интересно пронаблюдать в динамике движение точки по фазовой траектории.

```
Do[ParametricPlot[Evaluate[{x[t], x'[t]}/. nds], {t, 0, n},
    PlotRange → {{-3, 3}, {-3, 3}}, {n, 1, 50}] (в старых версиях)
Animate[ParametricPlot[Evaluate[{x[t], x'[t]}/. nds], {t, 0, n},
    PlotRange → {{-3, 3}, {-3, 3}}, PlotPoints → 1000 ], {n, 1, 50}]
```

Вот пример решения списка задач, отличающихся друг от друга начальным условием.

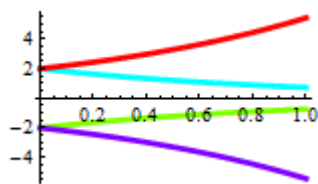
```
sol = Table[NDSolve[{x''[t] + 0.15x'[t] - x[t] + x[t]^3 == 0.3Cos[t],
                    x[0] == -1, x'[0] == a}, x, {t, 0, 25}],
    {a, 1, 1.1, 0.01}]
Plot[Evaluate[x[t]/. sol], {t, 0, 25}]
```



Встречаются уравнения с несколькими решениями. В следующем примере мы получаем 4 приближенных решения.

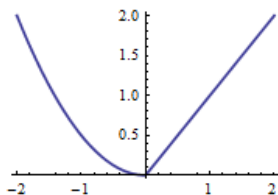
```
NDSolve[{y'[x]^2 - y[x]^2 == 0, y[0]^2 == 4}, y[x], {x, 0, 1}]
{{y[x] → InterpolatingFunction[{{0., 1.}}, " <> " ] [x]},
 {y[x] → InterpolatingFunction[{{0., 1.}}, "<>"] [x]},
 {y[x] → InterpolatingFunction[{{0., 1.}}, "<>"] [x]},
 {y[x] → InterpolatingFunction[{{0., 1.}}, " <> " ] [x]}}
```

**Plot[Evaluate[y[x]/. nds], {x, 0, 1}, PlotStyle
→ Table[{Thickness[0.02], Hue[i/4]}, {i, 1, 4}]]**

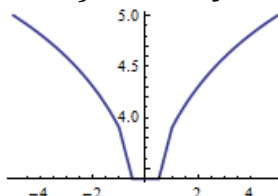


Функция **NDSolve** отлично решает кусочные ОДУ.

nds = NDSolve[{y'[x] == If[x < 0, x, 1], y[0] == 0}, y, {x, -2, 2}]
Plot[Evaluate[y[x]/. nds], {x, -2, 2}, PlotStyle → Thickness[0.01]]



**nds = NDSolve[{y'[x] == Which[
 $x < -1, \quad \frac{1}{x-1},$
 $x < -0.5, \quad -1,$
 $x < 0.5, \quad 0,$
 $x < 1, \quad 1,$
 $x \geq 1, \quad 1/(x+1)],$
 $y[-5] == 5\}, y, \{x, -5, 5\}]$
Plot[Evaluate[y[x]/. nds], {x, -5, 5}, PlotStyle → Thickness[0.01]]**



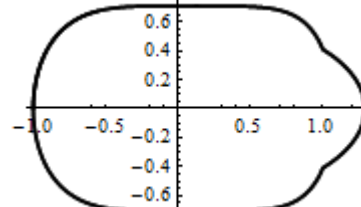
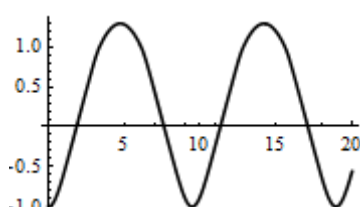
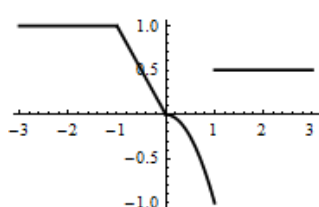
f[x_] = Piecewise[{{1, x < -1}, {-x, x < 0}, {-x^2, x < 1}}, 0.5];

Plot[f[x], {x, -3, 3}]

u = NDSolve[{x''[t] + f[x[t]]^2 x[t] == 0, x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]

Plot[x[t]/. u[[1, 1]], {t, 0, 20}]

**ParametricPlot[Evaluate[{x[t]/. u[[1, 1]], x'[t]/. u[[1, 1]]}, {t, 0, 20},
AspectRatio → Automatic, PlotPoints → 1000]**



На рисунке слева показана кусочная функция, в середине – график решения ОДУ, справа – фазовая траектория.

Иногда нужно остановить вычисления при достижении решением какого – либо значения или при выполнении какого – либо условия. Для этого в опциях функции `NDSolve` предусмотрено использование функции `WhenEvent`. Она имеет синтаксис

`WhenEvent[условие, действие]`

При включении этой функции в список опций `NDSolve`, на каждом шаге выполняется проверка логического выражения «условие» и, когда оно принимает значение `True`, происходит выполнение команды, представляющей «действие», при текущем значении независимой переменной. Эта команда может быть оператором подстановки, например, `y[x]→значение` или строкой `"StopIntegration"`, останавливающей вычисления. Допустимы и некоторые другие значения, с которыми вы можете познакомиться по справочной системе.

В следующем примере мы решаем ДУ до тех пор, пока решение не станет равным нулю (решением является `Cos[x]`).

```
nds = NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0,  

WhenEvent[y[x] == 0, "StopIntegration"]}, y, {x, 0, ∞}]  

{{y → InterpolatingFunction[{{0., 1.570796}}, " <> " ]}}  

xmax = nds[[1, 1, 2]][[1]][[1]][[2]]  

Plot[y[x]/. nds, {x, 0, xmax}]
```

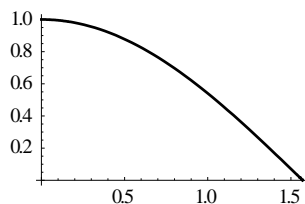


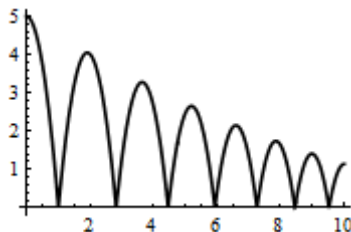
График решения построен до значения независимой переменной `x`, при котором решение обратилось в нуль. Для того, чтобы определить это значение, мы использовали индексацию.

Вот пример задачи, моделирующей движение «прыгающего» вертикально мяча (под действием силы притяжения), который теряет 10% своей скорости при отскоке от земли.

```
nds = NDSolve[{y''[t] == -9.81, y[0] == 5, y'[0] == 0,  

WhenEvent[y[t] == 0, y'[t] -> -0.9 y'[t]]}, y, {t, 0, 10}];  

Plot[y[t]/. nds, {t, 0, 10}]
```



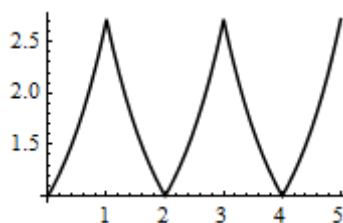
Здесь значение `y[t]` представляет вертикальную координату мяча, который начал падать с высоты 5 с нулевой начальной скоростью.

Вот пример использования этой опции для имитации кусочного ОДУ

```
sol = NDSolve[{y'[t] == a[t] y[t], y[0] == 1, a[0] == 1,  

WhenEvent[Mod[t, 1] == 0, a[t] -> -a[t]]}, y, {t, 0, 5}, DiscreteVariables -> a[t]];  

Plot[y[t]/. sol, {t, 0, 5}]
```

При численном решении уравнений важными являются следующие опции функции `NDSolve`, которые влияют на точность вычислений:

- `AccuracyGoal` определяет абсолютную погрешность вычислений на каждом шаге, используя количество значащих цифр; `AccuracyGoal` $\rightarrow \infty$ говорит, что эта опция не должна использоваться как критерий прекращения вычислений (т.е. будут использоваться другие опции);
- `PrecisionGoal` определяет «относительную погрешность» на каждом шаге вычислений и задается количеством значащих цифр; `PrecisionGoal` $\rightarrow \infty$ указывает, что эта опция не должна использоваться как критерий прекращения вычислений (будет использоваться опция `AccuracyGoal`);
- опция `WorkingPrecision` определяет количество значащих цифр, используемых во внутренних вычислениях; значение этой опции, как правило, должно быть больше значения опции `AccuracyGoal`; установка `WorkingPrecision` \rightarrow `MachinePrecision` приводит к вычислениям с процессорной точностью;
- опция `MaxSteps` задает максимальное количество шагов дискретизации;
- опция `MaxStepSize` задает максимальную длину шагов дискретизации; `MaxStepSize` \rightarrow `Infinity` отключает ограничение на размер шага;
- `StartingStepSize` задает начальный размер шага.

Для опции `PrecisionGoal` мы используем не совсем корректный термин «относительная погрешность». `PrecisionGoal` \rightarrow n определяет для значения x погрешность вычислений равную $|x|10^{-n}$. Когда включены опции `AccuracyGoal` \rightarrow m и `PrecisionGoal` \rightarrow n , то *Mathematica* пытается выполнять вычисления величины x с погрешностью не более $10^{-m} + |x|10^{-n}$.

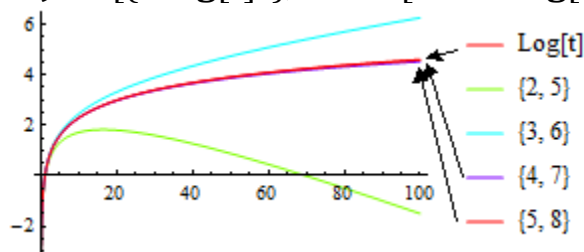
`NDSolve` подбирает размер шага так, чтобы ошибка вычислений не превосходила погрешностей, определяемых опциями `AccuracyGoal` и `PrecisionGoal`. Установка значения `Automatic` опций `AccuracyGoal` и `PrecisionGoal` эквивалентна значению `WorkingPrecision/2`.

Пример. Решим дифференциальное уравнение $y'' = -\frac{1}{t^2}$ на отрезке $[a; 100]$ с начальными условиями $y(a) = \ln(a)$, $y'(a) = 1/a$ при $a=0.001$. Его точное решение $y = \ln t$. Построим график точного решения $\ln t$ и приближенных, полученных при различных абсолютной и относительной погрешностях

```

a = 0.001;
ts = Table[
  NDSolveValue[{y''[t] == -1/t^2, y[a] == Log[a], y'[a] == 1/a}, y, {t, a, 100},
    AccuracyGoal → p - 3, PrecisionGoal → p], {p, 5, 8}];
Plot[Evaluate[Join[{Log[t], Table[ts[[i]][t], {i, 1, 4}]}], {t, a, 100},
  PlotStyle → Table[Hue[k], {k, 0, 1, 0.25}],
  PlotLegends → Join[{"Log[t]"}, Table[ToString[{p - 3, p}], {p, 5, 8}]]]

```



На предыдущем рисунке красной линией показан график точного решения и другими цветами графики приближенного решения при различных относительной и абсолютной погрешностях. Как видим, удовлетворительное приближение к точному решению получается при AccuracyGoal→4 и PrecisionGoal→7. Меньших значений недостаточно.

Пример. Отключим AccuracyGoal и решим ДУ, используя только, заданную по-умолчанию, относительную погрешность вычислений

```

a = 0.000001;
si = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}, AccuracyGoal → ∞]
eri[t_] = 1 - (x[t]/.si)[[1]]/(a Exp[t])
eri[1]
-8.00399 × 10-9

```

Точное решение уравнения известно $a \cdot e^{-t}$. Относительная погрешность в точке t определяется формулой $1 - \frac{x(t)}{a \cdot e^{-t}}$. Но нам еще пришлось выполнить замену $(x[t]/.si)[[1]]$.

Выполним те же вычисления, не отключая опцию AccuracyGoal.

```

ss = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}]
ers[t_] = 1 - (x[t]/.ss)[[1]]/(a Exp[t])
ers[1]
-0.000236

```

Без отключения опции AccuracyGoal относительная погрешность значительно больше. □

Пример. Отключим PrecisionGoal и решим ДУ, учитывая только абсолютную погрешность.

```

a = 10000;
si = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}, PrecisionGoal → ∞];
eri[t_] = a Exp[t] - x[t];
eri[1]/.si
{-4.84906 × 10-8}

```

Те же вычисления без отключения опции `PrecisionGoal`.

```
ss = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}];  
ers[t_] = aExp[t] - x[t];  
ers[1]/.ss  
{-0.000217574}
```

Точное решение этого уравнения известно $a \cdot e^{-t}$. Абсолютная ошибка в точке t определяется формулой $a \cdot e^{-t} - x(t)$. Без отключения опции `PrecisionGoal` абсолютная погрешность в точке $t=1$ больше.

□

Для повышения точности решения ОДУ используется опция `WorkingPrecision`. Ее значение обычно в два раза больше значений опций `PrecisionGoal` и `AccuracyGoal`.

Пример. Рассмотрим решение ОДУ 2-го порядка, точное решение которого известно – это функция `Cos[x]`. Решим его численно с различной точностью и сравним абсолютные погрешности в точке 2π , которые отобразим в таблице.

```
TableForm[Table[  
  s[i] = y/.First[NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0},  
    y, {x, 0, 2π}, WorkingPrecision → 4(1 + i)]];  
  {4(1 + i), Abs[1 - s[i][2π]], Length[s[i][[3, 1]]]},  
  {i, 0, 5}],  
  TableHeadings →  
    {None, {"WorkingPrecision", "Error", "Number of steps"}}]
```

WorkingPrecision	Error	Number of steps
4	0.008	15
8	0.0008358	27
12	0.00002980281	45
16	4.41292×10^{-8}	76
20	$4.223257269 \times 10^{-10}$	89
24	$1.551461146056 \times 10^{-11}$	120

Для пояснения 3-го столбца рассмотрим следующий набор точек

```
points = {{0, 0}, {1, 1}, {2, 3}, {3, 4}, {4, 3}, {5, 0}};
```

и выполним интерполяцию

```
ifun = Interpolation[points]  
InterpolatingFunction[{{0,5}}, "<> "]
```

При обращении к 3-му элементу выражения `InterpolatingFunction`

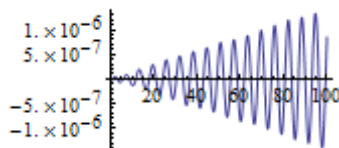
```
ifun[[3]]  
{{0,1,2,3,4,5}}
```

мы получаем список первых координат точек интерполяции. Функция `Length` возвращает количество элементов этого списка. В коде примера мы печатали это количество для каждого решения `s[i]` дифференциального уравнения, полученного при различных значениях опции `WorkingPrecision`.

Погрешность вычислений всегда растет с удалением от начальной точки

```
NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0}, y, {x, 0, 100}]
```

```
Plot[Evaluate[y[x] - Cos[x]/. %], {x, 0, 100}]
```



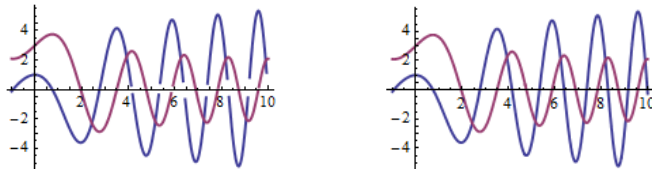
Отметим, что использование опции `WorkingPrecision` важно и в некоторых других случаях. Например, решим следующее ОДУ и построим график его решения (следующий рисунок слева)

```
sol = DSolve[{x'[t] + t y[t] == 0, 2 y'[t] - 3 x[t] == 0,
              x[0] == 1, y[0] == 3}, {x, y}, t];
```

```
Plot[Evaluate[{x[t], y[t]}/. sol], {t, -1, 10}]
```

Чтобы избавиться от «дыр» в графике мы увеличиваем точность внутренних вычислений (следующий рисунок справа).

```
Plot[Evaluate[{x[t], y[t]}/. sol], {t, -1, 10}, WorkingPrecision -> 20]
```



То же касается и фазовых траекторий этой задачи

```
ParametricPlot[Evaluate[{x[t], y[t]}/. sol], {t, -1, 10}]
```

```
ParametricPlot[Evaluate[{x[t], y[t]}/. sol], {t, -1, 10}, WorkingPrecision -> 20]
```



Пример. Рассмотрим уравнение.

```
nds = NDSolve[{y''[x] == -y[x], y[0] == 1, y'[0] == 0}, y, {x, 0, 500 Pi}]
```

Оно нормально решается и можно построить график его решения (Это функция `Cos[x]`). Но если увеличить интервал, т.е. решать его в интервале $\{x, 0, 500 \pi\}$, то мы получим сообщение

```
Maximum number of 10000 steps reached at the point x ==
1324.0481085415556`
```

Надо увеличить максимальное число шагов с помощью опции `MaxSteps`.

```
nds = NDSolve[{y''[x] == -y[x], y[0] == 1, y'[0] == 0}, y, {x, 0, 500 Pi},
              MaxSteps -> 20000]
```

Вот еще задача, в которой следует увеличить количество шагов

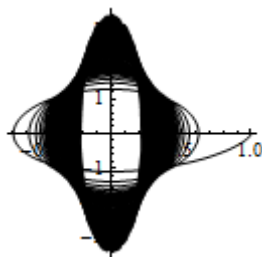
```
nds = NDSolve[{y''[x] + x y[x] == 0, y[0] == 1, y'[0] == 0}, y, {x, 0, 200}]
```

```
Maximum number of 10000 steps reached at the point x == 138.736390
```

Увеличим максимальное число шагов с помощью опции `MaxSteps`.

```
nds = NDSolve[{y''[x] + x y[x] == 0, y[0] == 1, y'[0] == 0},
              y, {x, 0, 200}, MaxSteps -> Infinity]
```

```
ParametricPlot[Evaluate[{y[x], y'[x]}/. nds], {x, 0, 200},
               AspectRatio -> 1, PlotStyle -> {Thickness[0.001], Black}]
```



NDSolve очень сложная функция. В ней реализовано много численных алгоритмов решения ОДУ различных типов. Обычно NDSolve сама выбирает метод численного решения. Однако иногда вы будете задавать метод с помощью опции Method. Мы не ставим перед собой задачу изложения всех методов, используемых этой функцией. С ними вы можете познакомиться по справочной системе.

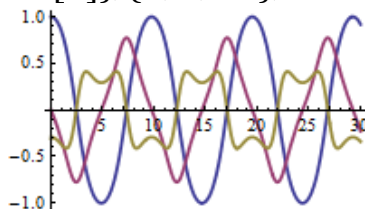
В версии Mathematica 9 добавлена новая функция NDSolveValue, которая в отличие от NDSolve возвращает решение не в форме правила подстановки, а в форме InterpolatingFunction функции.

```
ndsv = NDSolveValue[{y''[x] + Cos[y[x]]2 y[x] == 0, y[0] == 1,  
y'[0] == 0}, y, {x, 0, 30}]
```

```
InterpolatingFunction[{{0.,30.}}, " <> "]
```

Используя функцию ndsv, можно построить график решения, а также его первой и второй производных.

```
Plot[{ndsv[x], ndsv'[x], ndsv''[x]}, {x, 0, 30}, PlotStyle → Thickness[0.01]]
```

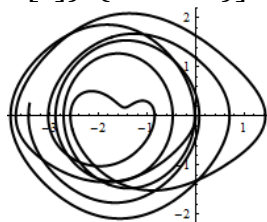


Вот как можно построить фазовую траекторию

```
ndsv = NDSolveValue[{x''[t] +  $\frac{x'[t]}{10}$  + Cos[x[t]] ==  $\frac{\text{Cos}[t]}{2}$ ,  
x[0] == x'[0] == 0}, x, {t, 0, 50}]
```

```
InterpolatingFunction[{{0.,50.}}, " <> "]
```

```
ParametricPlot[{ndsv[t], ndsv'[t]}, {t, 0, 50}]
```



Все возможности, которые мы продемонстрировали для функции NDSolve, имеются у функции NDSolveValue.

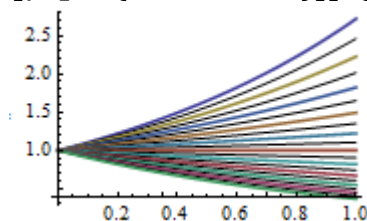
В версии Mathematica 9 появились две новые функции ParametricNDSolve и ParametricNDSolveValue. Они находят численное решение ОДУ с параметрами и отличаются одна от другой способом возврата решения.

Функция `ParametricNDSolve` возвращает решение в форме подстановки, и имеет следующий синтаксис

`ParametricNDSolve[уравнения, у, {x,xmin,xmax}, параметры]`

Она находит численное решение *уравнений с параметрами* относительно неизвестной функции y с независимой переменной x , изменяемой в диапазоне от x_{\min} до x_{\max} .

```
ps = ParametricNDSolve[{y'[t] + a y[t] == 0, y[0] == 1}, {y}, {t, 0, 1}, {a}]
{y → ParametricFunction[" < "" > "]}
Plot[Evaluate[Table[y[a][t]/. ps, {a, -1, 1, .1}], {t, 0, 1}]
```



Результат возвращается в форме подстановки для искомой функции y `ParametricFunction` объекта. После подстановки вместо параметра конкретного значения этот объект становится `InterpolatingFunction` объектом и, фактически, функцией относительно независимой переменной

```
y1 = y[1]/. ps
```

```
InterpolatingFunction[{{0.,1.}}, " <> "]
```

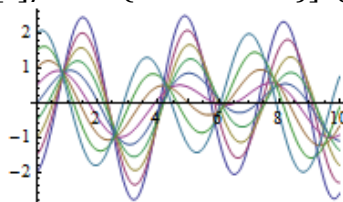
```
y1[0.5]
```

```
0.367879
```

При обращении к решению с параметром мы получаем «функцию двух переменных», первой из которых является параметр задачи, а второй – независимая переменная. Например, `y[1][0.5]/.ps` возвращает значение в точке $t=0.5$ решения задачи с параметром $a=1$.

Параметров в уравнении может быть несколько

```
sol = ParametricNDSolve[{y''[t] + 4 y[t] == Cos[t] + Sin[y[t]],
                        y[0] == a, y'[0] == b}, y, {t, 0, 10}, {a, b}]
Plot[Evaluate@Table[y[a, 1][t]/. sol, {a, -2, 2, .5}], {t, 0, 10}, PlotRange → All]
```



Функция `ParametricNDSolveValue` возвращает решение в форме функции, и имеет следующий синтаксис

`ParametricNDSolveValue[уравнения, выражение, {x,xmin,xmax}, параметры]`

Она возвращает в форме функции значение *выражения*, численно решая *уравнения с параметрами* и с независимой переменной x , изменяющейся в диапазоне от x_{\min} до x_{\max} . В качестве *выражения* чаще всего выступает имя искомой функции.

```
ps = ParametricNDSolveValue[{y''[t] + a y[t] == 0, y[0] == 1,
                            y'[0] == 0}, y, {t, 0, 10}, {a}]
```

```
ParametricFunction[" < "" > "]
```

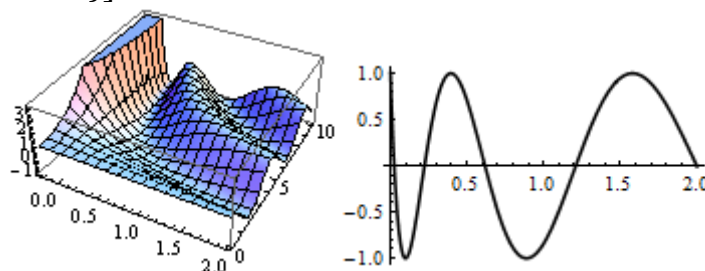

Теперь `ps` является функцией. Мы определили параметрическую функцию (переменной t), зависящую от параметра a .

```
ps[2][0.5]  
0.760245
```

Ее следует использовать так, как она была создана – каждый аргумент в своих квадратных скобках. По-другому она работать не будет. Возможные варианты ее использования приведены ниже

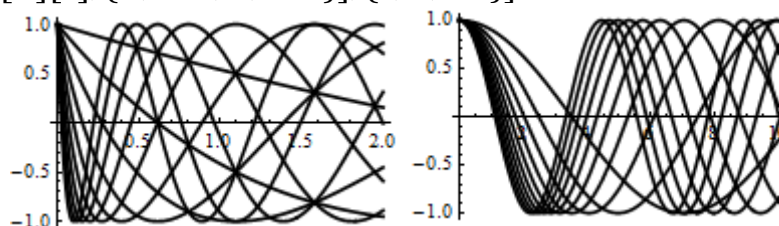
```
Plot3D[ps[a][t], {a, -0.25, 2}, {t, 0, 10}]
```

```
Plot[ps[a][10], {a, 0, 2}]
```



```
Plot[Table[ps[a][t], {t, 1, 10}], {a, 0, 2}]
```

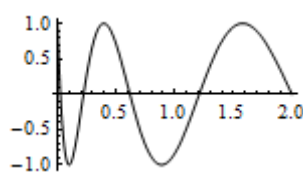
```
Plot[Table[ps[a][t], {a, 0.2, 2, 0.2}], {t, 0, 10}]
```



Можно находить решение, зависящее от параметра, при фиксированном значении переменной t . Например

```
pfun = ParametricNDSolveValue[{y''[t] + a y[t] == 0, y[0] == 1,  
y'[0] == 0}, y[10], {t, 0, 10}, {a}]
```

```
Plot[pfun[a], {a, 0, 2}]
```



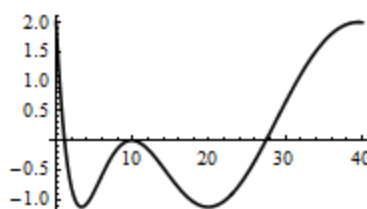
Фактически, это тот же график, который мы построили выше командой `Plot[ps[a][10], {a, 0, 2}]`.

Функция `ParametricNDSolveValue` возвращает не обязательно решение. Это может быть некоторое выражение, получаемое из искомого решения.

```
eqns = {y''[t] + a y[t] == 0, y[0] == 1, y'[0] == 0};
```

```
fun12 = ParametricNDSolveValue[eqns, y[1] + y[2], {t, 0, 15}, {a}]
```

```
Plot[fun12[a], {a, 0, 40}]
```



Проверим, что это то же, что $y[a][1] + y[a][2]$

```
fun = ParametricNDSolveValue[eqns, y, {t, 0, 15}, {a}]
```

```
Plot[fun[a][1] + fun[a][2], {a, 0, 40}]
```

Мы получим тот же график, что и выше.

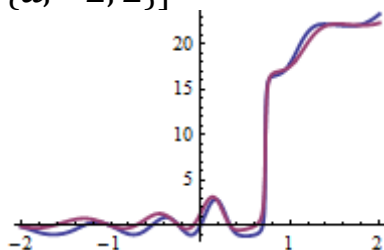
Вот еще пример.

```
eqns = {y''[t] + y[t] == 3aSin[y[t]], y[0] == y'[0] == 1};
```

```
pfun1 = ParametricNDSolveValue[eqns, Sum[y[i], {i, 1, 10}], {t, 0, 15}, {a}];
```

```
pfun2 = ParametricNDSolveValue[eqns, Integrate[y[s], {s, 0, 10}], {t, 0, 15}, {a}];
```

```
Plot[{pfun1[a], pfun2[a]}, {a, -2, 2}]
```



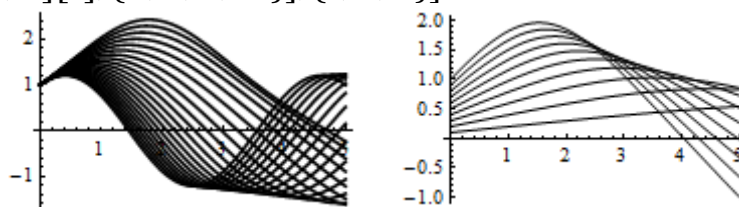
Как мы говорили ранее, параметров в уравнении может быть несколько

```
ps = ParametricNDSolveValue[{y''[t] + y[t] == aSin[y[t]],
```

```
y[0] == y'[0] == b}, y, {t, 0, 5}, {a, b}]
```

```
Plot[Table[ps[a, 1][t], {a, -1.5, 1.5, .15}], {t, 0, 5}, PlotRange -> All]
```

```
Plot[Table[ps[1, b][t], {b, 0, 1, .1}], {t, 0, 5}]
```



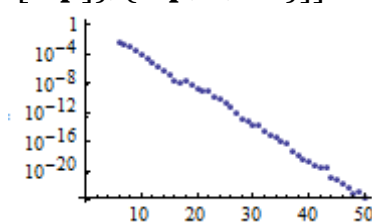
Функция `ParametricNDSolveValue` имеет такие же опции как и функция `NDSolve`.

Вот еще один пример ее использования. Численно решим задачу $x'(t) = x(t)$, $x(0) = 1$ для различных значений `WorkingPrecision` и графически изобразим абсолютную погрешность вычислений в точке $t=1$, равную $|x(1) - e|$

```
err = ParametricNDSolveValue[{x'[t] == x[t], x[0] == 1}, Abs[x[1] - E],
```

```
{t, 0, 1}, {wp}, WorkingPrecision -> wp];
```

```
ListLogPlot[Table[{wp, err[wp]}, {wp, 6, 50}]]
```



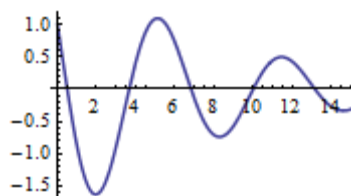
Пример. Найдем начальное условие $x'(0)$ при котором решение $x(t)$ ДУ обратиться в 0 при $t=10$, т.е. $x(10) = 0$


```

pf = ParametricNDSolveValue[{x''[t] + 0.25 x'[t] + x[t] == 0,
                             x[0] == 1, x'[0] == s}, x, {t, 0, 15}, {s}];
root = FindRoot[pf[s][10] == 0, {s, 11}]
{s -> -1.95507}

```

Для проверки построим график решения при этом значении параметра **Plot[pf[s/.root][t], {t, 0, 15}, PlotRange -> All]**

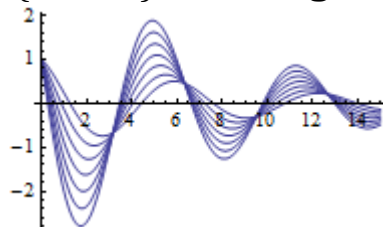


Сравним поведение кривых при близких значениях параметра s.

```

s0 = s/.root;
sL = Table[s0 + i * 0.5, {i, -3, 3}]
{-3.455, -2.955, -2.455, -1.95507, -1.455, -0.955, -0.455}
Plot[Table[pf[s][t], {s, sL}], {t, 0, 15}, PlotRange -> All]

```



4.2.2 Численное решение систем ОДУ

Системы ОДУ численно решаются с помощью тех же функций, которые используются для решения одного ОДУ. Начиная с «древних версий» системы *Mathematica* для этого используется функция `NDSolve`.

```

sol = NDSolve[{x'[t] == y[t], y'[t] == -0.01 y[t] - Sin[x[t]],
               x[0] == 0, y[0] == 2.1}, {x, y}, {t, 0, 1}]
{{x -> InterpolatingFunction[{{0., 1.}}, " <> "],
  y -> InterpolatingFunction[{{0., 1.}}, " <> "]]}

```

Обратите внимание что, мы имеем единственное решение в виде двух правил подстановки для x и y соответственно. Чтобы получить из них две функции достаточно выполнить следующие команды

```

x = x/.sol[[1]]
y = y/.sol[[1]]
InterpolatingFunction[{{0., 100.}}, " <> "]
InterpolatingFunction[{{0., 100.}}, " <> "]

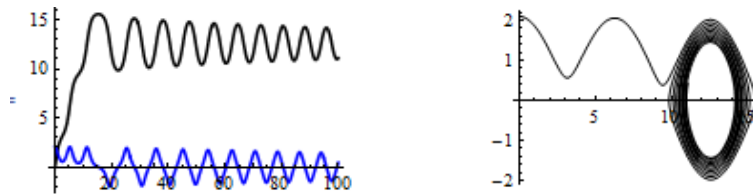
```

Как видно из последнего решения, x и y являются уже функциями и для них можно применять обычные математические действия и строить график.

```

Plot[{x[t], y[t]}, {t, 0, 100}, PlotStyle ->
  {{Thickness[0.01], RGBColor[0, 0, 0]}, {Thickness[0.01], RGBColor[0, 0, 1]}}]
ParametricPlot[{x[t], y[t]}, {t, 0, 100}, PlotStyle ->
  {RGBColor[0, 0, 0]}, AspectRatio -> 0.7]

```

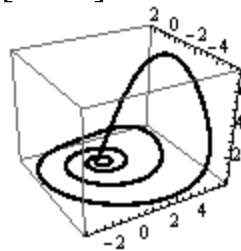


Вот пример построения фазовой траектории системы 3 – х ОДУ без создания функций решения

Remove[*x, y, z, t*]

```
sol = NDSolve[{x'[t] == -y[t] - z[t], x[0] == -0.04,
               y'[t] == x[t] + 0.425y[t], y[0] == -0.3,
               z'[t] == 2 - z[t](4 - x[t]), z[0] == 0.52},
               {x, y, z}, {t, 0, 25}]
```

```
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. sol], {t, 0, 25},
                  PlotStyle → {Thickness[0.01], RGBColor[0, 0, 0]}, PlotRange → All]
```



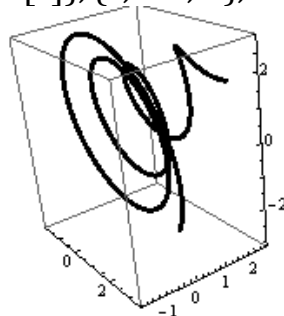
Вот пример построения фазовой траектории системы 3 – х ОДУ с созданием функций решения

Remove[*x, y, z, t*]

```
nds = NDSolve[{x'[t] == y[t] - z[t], y'[t] == z[t] - x[t], z'[t] =
               = x[t] - 2y[t], x[0] == 1, y[0] == 0, z[0] =
               = 2}, {x, y, z}, {t, -4, 8}]
```

```
x = nds[[1, 1, 2]]; y = nds[[1, 2, 2]]; z = nds[[1, 3, 2]]
```

```
ParametricPlot3D[{x[t], y[t], z[t]}, {t, -4, 8}, PlotStyle → Thickness[0.01]]
```

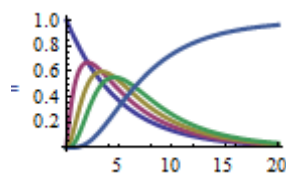


Неизвестные функции задачи не обязаны быть представлены уникальными идентификаторами (именами). Когда неизвестных функций много вам, вероятно, будет удобно обозначать их, например так, *y*[*i*].

```
eqns = Join[Table[y[i]'[x] == y[i - 1][x] - y[i][x], {i, 2, 4},
             {y[1]'[x] == -0.2y[1][x], y[5]'[x] == 0.2y[4][x],
             y[1][0] == 1}, Table[y[i][0] == 0, {i, 2, 5}]];
```

```
nds = NDSolve[eqns, Table[y[i], {i, 5}], {x, 0, 20};
```

```
Plot[Evaluate[Table[y[i][x], {i, 1, 5}] /. nds], {x, 0, 20}]
```



Функция `NDSolve` может рассматривать обозначение $y[x]$ как обозначение **вектор – функции**, если для $y[x]$ представить вектор инициализаций, например, $y[0] == \{v_1, v_2, \dots, v_n\}$. Для обращения к компонентам такой вектор функции следует выбирать выражения $y[x][[1]]$, $y[x][[2]]$,...

Пример. Одну и ту же задачу о колебании линейного осциллятора можно записать и решить в *Mathematica* тремя способами.

1. Как систему ОДУ 1 – го порядка

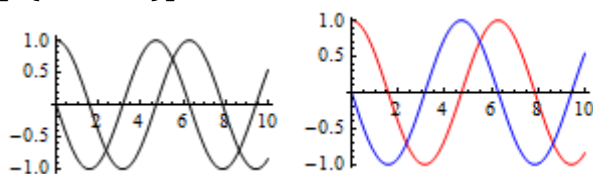
```
NDSolve[{x'[t] == y[t], y'[t] == -x[t], x[0] == 1,
          y[0] == 0}, {x, y}, {t, 0, 10}]
```

2. Как ОДУ 2 – го порядка

```
NDSolve{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0}, y, {x, 0, 10}]
```

3. Как задачу для вектор функции

```
nds = NDSolve{z'[t] == {{0, 1}, {-1, 0}}.z[t], z[0] == {1, 0}}, z, {t, 0, 10}]
Plot[z[t]/.First[nds], {t, 0, 10}]
```



Чтобы обратиться к каждой компоненте вектор функции как к отдельной функции можно выполнить команды

```
g = nds[[1, 1, 2]]
Plot[{g[x][[1]], g[x][[2]]}, {x, 0, 10}, PlotStyle → {{Red}, {Blue}}]
```

График, построенный последней командой, показан на предыдущем рисунке справа. Обратите внимание, что при таком построении вы можете каждой кривой задать свой стиль/цвет.

□

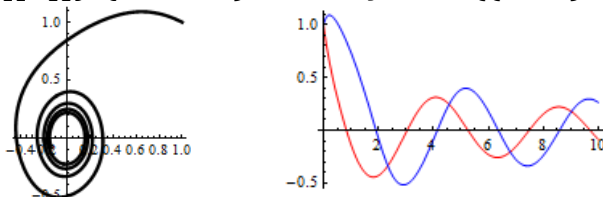
Пример. Иногда удастся записать нелинейную систему ОДУ как задачу относительно вектор – функции. При этом, например, запись $x[t]^3$ будет означать покомпонентное возведение в степень, а запись $\{\{0, -1\}, \{2, 0\}\}.x[t]$ – «матричное» умножение (функция `Dot` – точка).

```
s = NDSolve{x'[t] == {{0, -1}, {2, 0}}.x[t] - x[t]^3, x[0] == {1, 1}}, x, {t, 20}]
```

```
ParametricPlot[Evaluate[{x[t]}/.s], {t, 0, 20}]
```

```
g = s[[1, 1, 2]]
```

```
Plot[{g[x][[1]], g[x][[2]]}, {x, 0, 10}, PlotStyle → {{Red}, {Blue}}]
```



Та же задача может быть записана как обычная система ОДУ 1 – го порядка

```
s = NDSolve[{x'[t] == -y[t] - x[t]^3, y'[t] == 2x[t] - y[t]^3,
             x[0] == y[0] == 1}, {x, y}, {t, 20}];
```

□

Вот система ОДУ 2 – го порядка, записанная относительно вектор – функции

```
s = NDSolve[{y''[x] +  $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 2 \\ 1 & 1 & 3 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix} \cdot y[x] == 0,$ 
             y[0] == y'[0] == {1, 1, 1, 1}}, y, {x, 0, 8}];
```

```
{y → InterpolatingFunction[{{0., 8.}}, " <> "]}
```

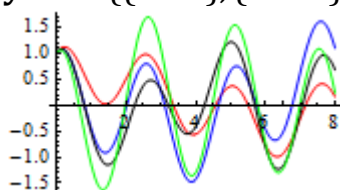
```
y[5]/.s
```

```
{{0.36310, 0.658461, 1.215740, 1.4692304}}
```

```
g = s[[1, 1, 2]]
```

```
Plot[{g[x][[1]], g[x][[2]], g[x][[3]], g[x][[4]]}, {x, 0, 8},
```

```
PlotStyle → {{Red}, {Blue}, {Black}, {Green}}]
```



В следующем примере мы строим несколько фазовых траекторий системы ДУ, соответствующих разным начальным условиям, используя функцию ParametricNDSolve.

```
Remove[x, y, z, t, X, Y, Z]
```

```
nds = ParametricNDSolve[{x'[t] == -x[t] + z[t], x[0] == 0.1a,
```

```
y'[t] == -y[t] - z[t], y[0] == 1,
```

```
z'[t] == y[t] - z[t], z[0] == 0},
```

```
{x, y, z}, {t, 0, π}, {a}]
```

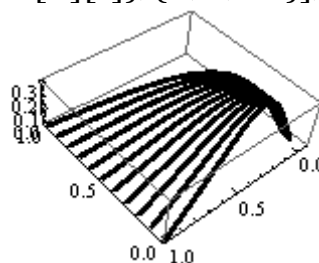
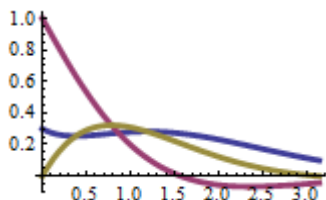
```
X[a_] = x[a]/.nds
```

```
Y[a_] = y[a]/.nds
```

```
Z[a_] = z[a]/.nds
```

```
Plot[{X[3][t], Y[3][t], Z[3][t]}, {t, 0, π}]
```

```
ParametricPlot3D[Table[{X[a][t], Y[a][t], Z[a][t]}, {a, 0, 10}], {t, 0, π}]
```



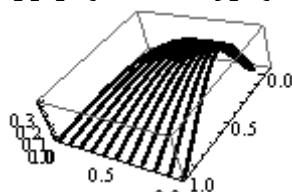
Пример. Функция ParametricNDSolve также может работать с вектор – функциями.

```
s = ParametricNDSolve[{x'[t] == {{-1, 0, 1}, {0, -1, -1}, {0, 1, -1}}.x[t],
                     x[0] == {0.1a, 1, 0}}, x, {t, 0, π}, {a}]
```

```
{x → ParametricFunction[" < "" > "]}
```

$g = s[[1, 2]]$

ParametricPlot3D[Table[g[a][t], {a, 0, 10}], {t, 0, π }]



Соответствующую систему ОДУ 1 – го порядка относительно 3 – х неизвестных функций мы решали ранее в этом параграфе.

Также как и для одного ОДУ для решения систем ОДУ в Mathematica 9 можно использовать функцию **NDSolveValue**.

Пример. Исследуем решение задачи Коши для системы уравнений

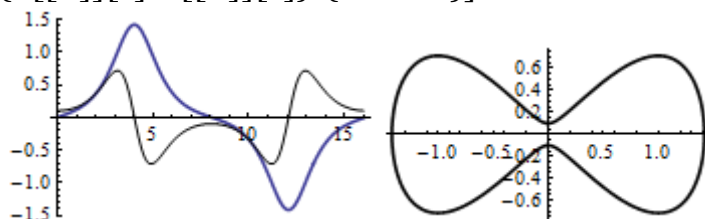
$$x' = y(t), \quad y' = -x^3(t) + x(t), \quad x(0) = 0, \quad y(0) = 0.1$$

Имеем

s = NDSolveValue[{ $x'[t] == y[t]$, $y'[t] == x[t] - x[t]^3$,
 $x[0] == 0$, $y[0] == 0.1$ }, {x, y}, {t, 0, 16}]

Plot[{s[[1]][t], s[[2]][t]}, {t, 0, 16}]

ParametricPlot[{s[[1]][t], s[[2]][t]}, {t, 0, 16}]



На левом рисунке показаны графики функций $x(t)$, $y(t)$, а на правом – фазовая траектория.

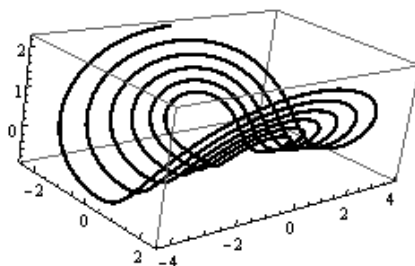
Пример. Решим систему дифференциальных уравнений

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = 0.1y - x \cdot (z - 1) - x^3, \quad \frac{dz}{dt} = x \cdot y - 0.1z$$

с начальными условиями $x(0) = 1$, $y(0) = 1$, $z(0) = 0$ и построим ее фазовый портрет.

s = NDSolveValue[{ $x'[t] == y[t]$, $y'[t] == 0.1y[t] - x[t](z[t] - 1) - x[t]^3$,
 $z'[t] == x[t]y[t] - 0.1z[t]$, $x[0] == 1$, $y[0] == 1$, $z[0] == 0$ },
{x, y, z}, {t, 0, 25}]

ParametricPlot3D[{s[[1]][t], s[[2]][t], s[[3]][t]}, {t, 0, 25}]



Пример. Решим систему уравнений

$$\begin{cases} x' = s \cdot (y - x) \\ y' = x \cdot (r - z) - y, \\ z' = x \cdot y - b \cdot z \end{cases}$$

где s, r, b некоторые параметры.

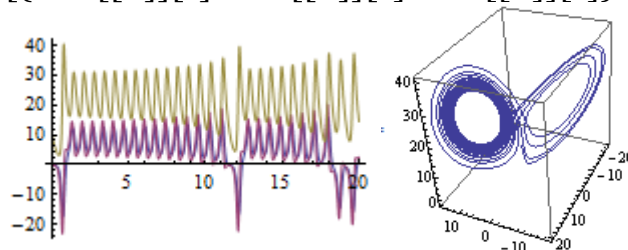
$s = 10; r = 25; b = 3;$

$\text{sys} = \{x'[t] == s(y[t] - x[t]), y'[t] == x[t](r - z[t]) - y[t],$
 $z'[t] == x[t]y[t] - bz[t], x[0] == 1, y[0] == -1, z[0] == 10\};$

$\text{nds} = \text{NDSolveValue}[\text{sys}, \{x, y, z\}, \{t, 0, 20\}]$

$\text{Plot}[\{\text{nds}[[1]][t], \text{nds}[[2]][t], \text{nds}[[3]][t]\}, \{t, 0, 20\}]$

$\text{ParametricPlot3D}[\{\text{nds}[[1]][t], \text{nds}[[2]][t], \text{nds}[[3]][t]\}, \{t, 0, 20\}]$



Влияние параметров s, r, b можно исследовать по поведению фазовых траекторий в блоке `Manipulate`.

Manipulate[

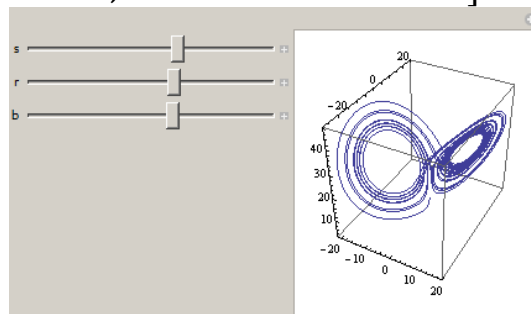
$\text{sys} = \{x'[t] == s(y[t] - x[t]), y'[t] == x[t](r - z[t]) - y[t],$
 $z'[t] == x[t]y[t] - bz[t], x[0] == 1, y[0] == -1, z[0] == 10\};$

$\text{nds} = \text{NDSolveValue}[\text{sys}, \{x, y, z\}, \{t, 0, 20\}];$

$\text{ParametricPlot3D}[\{\text{nds}[[1]][t], \text{nds}[[2]][t], \text{nds}[[3]][t]\}, \{t, 0, 20\},$
 $\text{PlotRange} \rightarrow \text{All}],$

$\{\{s, 10\}, 1, 20\}, \{\{r, 25\}, 5, 45\}, \{\{b, 3\}, 1, 5\},$

$\text{ControlPlacement} \rightarrow \text{Left}, \text{AutoAction} \rightarrow \text{False}]$



□

Обычно функция `NDSolve` сама выбирает метод численного решения. В справочной системе *Mathematica* приводится уравнение Ван-дер-Поля $z'' + z - K \cdot (1 - z^2) \cdot z' = 0$, решение которого обычными методами типа Рунге – Кутта дает неудовлетворительный результат. Запишем уравнение в виде системы (полагаем $K=1000$)

$$\begin{cases} x' = y \\ y' = -x + K \cdot (1 - x^2) \cdot y \end{cases}, \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

```
vdp = { $x'[t] == y[t]$ ,  $y'[t] == -x[t] + 1000(1 - x[t]^2) \cdot y[t]$ ,  

 $x[0] == 2, y[0] == 0$ };
```

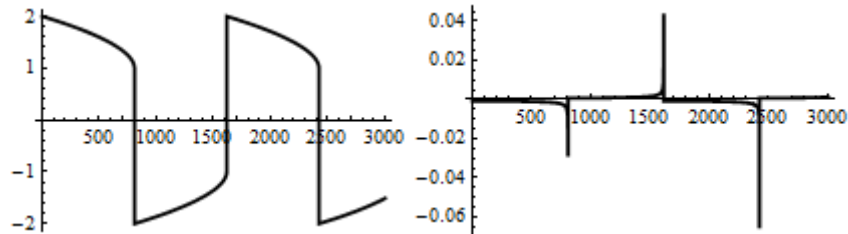
```
sol = NDSolve[vdp, {x, y}, {t, 3000}]  

{{x → InterpolatingFunction[[{0., 3000.}], " <> "],  

y → InterpolatingFunction[[{0., 3000.}], " <> "]]}  

Plot[Evaluate[x[t]/.sol], {t, 0, 3000}, PlotRange → All  

Plot[Evaluate[y[t]/.sol], {t, 0, 3000}, PlotRange → All
```



Система *Mathematica* сама правильно выбрала метод решения. Дело в том, что эта задача является примером так называемых жестких систем, для решения которых в *Mathematica* используются специальные методы решения. Методы не предназначенные для таких задач, вероятно, не построят решение

```
NDSolve[{ $x'[t] == y[t]$ ,  $y'[t] == -x[t] + 1000(1 - x[t]^2)y[t]$ ,  

 $x[0] == 2, y[0] == 0$ }, {x, y}, {t, 2000}, Method → "ExplicitRungeKutta"]  

NDSolve::ndstf: At t==0.008522, system appears to be  

stiff. Methods Automatic, BDF, or StiffnessSwitching may  

be more appropriate.
```

4.3 Представление решений ОДУ в системе *Mathematica*

Наглядное графическое представление решений ОДУ и их систем является существенным элементом исследования. В предыдущих параграфах этого пособия мы показали как строить графики решений, фазовые траектории и анимацию. Здесь мы приведем примеры использования других средств системы *Mathematica*, которые также помогут быть полезны при изучении поведения решений ОДУ.

1⁰. Напомним об опциях *Epilog* и *Prolog*, которые имеют функции построения графиков. Их значениями могут быть графические примитивы, которые отображаются в области графика после или до его создания.

В следующем примере вместе с графиком решений мы отображаем точки дополнительных условий.

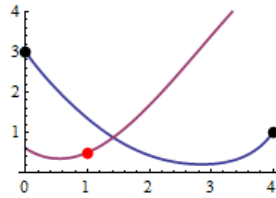
```
sol = NDSolve[{ $x''[t] == y[t]x[t]$ ,  $y'[t] == 2 - x[t]$ ,  

 $x[0] == 3, x[4] == 1, y[1] == 0.5$ }, {x, y, t}]  

Plot[Evaluate[{x[t], y[t]}/.sol[[1]]], {t, 0, 4},  

Epilog → {PointSize[Large], Point[{0, 3}], Point[{4, 1}], Red, Point[{1, 0.5}]},  

PlotRange → {{-0.1, 4.1}, {0, 4}}, PlotStyle → Thickness[0.01]
```

Примитив, используемый в `Epilog`, можно менять/перемещать вместе с параметром анимации.

Remove[*x*, *y*]

```
sol = NDSolve[{x''[t] == y[t]x[t], y'[t] == 2 - x[t],
               x[0] == 3, x[4] == 1, y[1] == 0.5}, {x, y}, t]
```

```
x = sol[[1, 1, 2]]; y = sol[[1, 2, 2]];
```

Animate[

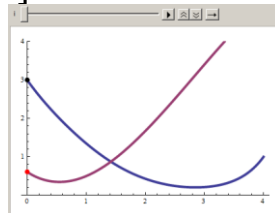
```
  Plot[Evaluate[{x[t], y[t]}], {t, 0, 4},
```

```
    PlotStyle → Thickness[0.01], PlotRange → {{-0.1, 4.1}, {0, 4}},
```

```
    Epilog → {PointSize[Large], Point[{i, x[i]}],
```

```
              Red, Point[{i, y[i]}]}, {i, 0, 4},
```

```
  AnimationRunning → False]
```



2⁰. Практически любой управляющий объект (`Control`) можно использовать для улучшения наглядности представления решений ОДУ и их систем. Здесь мы продемонстрируем использование некоторых из них.

Функция `LocatorPane` предоставляет область с «локатором», который можно перемещать с помощью мыши. В формате

`LocatorPane[Dynamic[pt], тело]`

она позволяет динамически менять положение указателя и использовать его координаты в теле (последовательности команд).

В следующем примере мы создаем 3 локатора, координаты которых используются для построения трех фазовых траекторий, получаемых при трех различных начальных условиях.

```
xmax = 20;
```

```
DynamicModule[{pt = {{-1, 0}, {0, 1}, {-2, 0}}, dd, p, pp},
```

```
  LocatorPane[Dynamic[pt],
```

```
    Dynamic[
```

```
      pp = Table[
```

```
        dd = DSolve[{y''[x] + 0.3y'[x] + y[x] == 1,
```

```
                    y[0] == pt[[i, 1]], y'[0] == pt[[i, 2]]}, y, x];
```

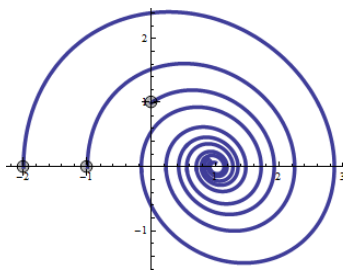
```
        p = ParametricPlot[Evaluate[{y[x]/.dd[[1]],
```

```
                                y'[x]/.dd[[1]]}], {x, 0, xmax},
```

```
                                PlotRange → {{-3, 3}, {-3, 3}}];
```

```
        p, {i, 1, 3}];
```

```
      Show[pp]]]]]
```

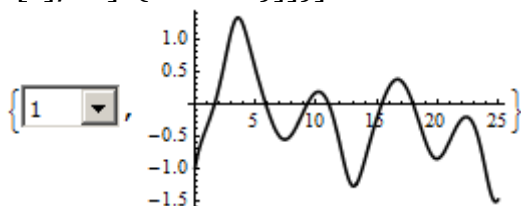



«Захватите» мышью «локатор» и перемещайте его. Каждое новое положение «локатора» определяет новые начальные значения и, следовательно, новую фазовую траекторию системы. Кривые будут динамически перерисовываться.

□


С помощью элемента «выпадающий список» (создается функцией `PopupMenu`) можно менять значение какого-либо параметра в ДУ и автоматически получать новое решение.

```
DynamicModule[{a},
  {PopupMenu[Dynamic[a],{1 → 1, 2 → 1.25, 3 → 1.5, 4 → 1.75, 5 → 2}],
  Dynamic[
    s = NDSolve[{x''[t] + x'[t] - x[t] + x[t]^3 == Sin[t],
      x[0] == -1, x'[0] == a}, x, {t, 0, 25}];
    Plot[Evaluate[x[t]/.s], {t, 0, 25}]]]
```

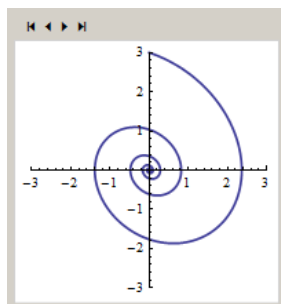


Выбор в списке значения автоматически перестраивает график решения.

□

Элемент `SlideView` (функция, которая его создает) принимает список объектов и представляет один из них в документе. Щелчки по кнопкам  элемента позволяют переходить последовательно от представления одного объекта к другому. Отображаемым объектом может быть, например, график решения или фазовая траектория.

```
SlideView[
  Table[
    ds = DSolve[{x'[t] == y[t], y'[t] == -x[t] - y[t]/a,
      x[0] == 0, y[0] == a}, {x, y}, t];
    p = ParametricPlot[
      Evaluate[{x[t]/.ds[[1, 1]], y[t]/.ds[[1, 2]]}], {t, 0, 100},
      PlotRange → {{-3, 3}, {-3, 3}}, PlotStyle → Thickness[0.01];
    p, {a, 1, 4}]]
```

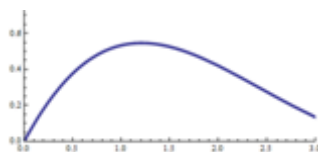


□

Элемент `FlipView` (одноименная функция, которая его создает) тоже может представлять один объект из нескольких, заданных в списке. Только переход от одного объекта к другому выполняется при щелчке мышью по объекту, например, по графику решения.

```
ds = DSolve[{x'[t] == y[t], y'[t] == -x[t] - y[t],  
             x[0] == 0, y[0] == 1}, {x, y}, t];
```

```
FlipView[  
  {ParametricPlot[Evaluate[{x[t]/.ds[[1,1]], y[t]/.ds[[1,2]]}, {t, 0, 100},  
    PlotRange -> {{-1, 1}, {-1, 1}},  
    Plot[Evaluate[x[t]/.ds[[1,1]], {t, 0, 100}, PlotRange -> {{0, 3}, {0, 1}},  
    Plot[Evaluate[y[t]/.ds[[1,2]], {t, 0, 100}, PlotRange -> {{0, 3}, {-1, 1}},  
  }]
```

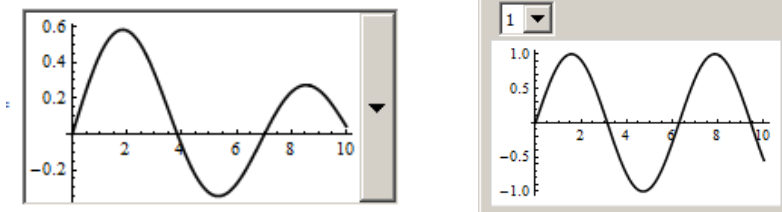


Щелкая мышью по области рисунка вы можете переходить последовательно от одного графика к другому.

□

Аналогичным свойством обладает элемент `PopupView`, `MenuView`. Они принимают список объектов, например, графических и по очереди отображают их в своей рабочей области.

```
PopupView[Table[Plot[Bessel[n, x], {x, 0, 10}], {n, 3}]]  
MenuView[Table[Plot[Sin[nx], {x, 0, 10}], {n, 5}]]
```



Первый из этих элементов показан на предыдущем рисунке слева, второй – справа. Внутри списков, генерируемых функцией `Table`, вы можете поставить любые, имеющие экранное представление функции, например, графики решений ОДУ или таблицы значений решений.

□

Однако, наиболее часто используемым управляющим объектом, вероятно, является `Manipulate`.

Remove[`z, x, y`]

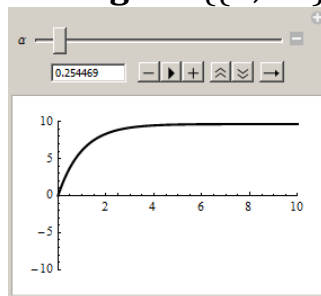
sol = **DSolve**[{**D**[`x[t, α]`, {`t`, 2}] == -**D**[`x[t, α]`, `t`],
`x[0, α]` == 0, **Derivative**[1, 0][`x`][0, α] == 10**Cos**[α]}, `x[t, α]`, `t`]

`z[t_, α_] = sol[[1, 1, 2]]`

{{`x[t, α]` → 10 $e^{-t}(-1 + e^t)$ **Cos**[α]}}

Manipulate[

Plot[`z[t, α]`, {`t`, 0, 10}, **PlotRange** → {{0, 10}, {-10, 10}}, {α, 0, π}]



Перемещая бегунок манипулятора, мы меняем значение параметра α и, тем самым, строим график решения $z(t, \alpha)$ при другом α . Раскрыв значок «+» справа от бегунка, можно наблюдать за значением параметра α .

□

3⁰. Кроме графиков решений и фазовых траекторий существуют другие способы улучшенного наглядного представления решений ОДУ. Так фазовые траектории иногда удобно рисовать поверх векторных полей, которые в системе *Mathematica* можно строить различными способами. Напомним некоторые понятия, связанные с векторными полями на плоскости.

Если каждой точке (x, y) двумерного пространства поставить в соответствие вектор $\mathbf{v}(x, y) = (P(x, y), Q(x, y))$, в результате получим векторное поле. Один из способов его визуализации состоит в рисовании в некотором множестве точек плоскости стрелок, представляющих в некотором масштабе значение вектора $\mathbf{v}(x, y)$ в этих точках. Такой рисунок в *Mathematica* создается функцией `VectorPlot`, которой в качестве аргументов передаются скалярные функции $P(x, y), Q(x, y)$. Кроме того, для наглядного представления векторных полей используются линии тока (векторные линии, силовые линии). Через каждую точку плоскости проходит одна линия. За исключением точек, где поле не определено или $\mathbf{v}(x, y) = (0, 0)$, линии тока никогда не пересекаются. В декартовых координатах дифференциальные уравнения линий тока имеют вид $\frac{dx}{P(x, y)} = \frac{dy}{Q(x, y)}$. Поле линий тока в *Mathematica* строится функцией `StreamPlot`.

Пример. Построим несколько фазовых траекторий системы двух ДУ и векторное поле, соответствующее этой системе.

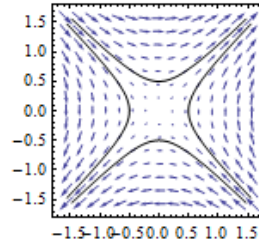
sys = {`x'[t]` == `y[t]`, `y'[t]` == `x[t]`};

bc = {{`x[0]` == 0, `y[0]` == 0.5}, {`x[0]` == 0.5, `y[0]` == 0},
{`x[0]` == 0, `y[0]` == -0.5}, {`x[0]` == -0.5, `y[0]` == 0}};

```

pp = Table[
    se = Join[sys, bc[[i]]];
    u = DSolve[se, {x, y}, t];
    p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, -1.8, 1.8};
    p, {i, 1, 4}];
vp = VectorPlot[{y, x}, {x, - $\frac{\pi}{2}$ ,  $\frac{\pi}{2}$ }, {y, - $\frac{\pi}{2}$ ,  $\frac{\pi}{2}$ };
Show[vp, pp]

```

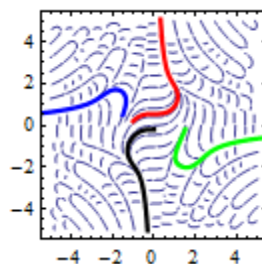


Пример. В следующем примере мы строим поле линий тока векторного поля и некоторые из таких линий, которые получаются из решения соответствующей системы ОДУ при различных начальных условиях.

```

sys = {x'[t] == y[t]Cos[x[t]y[t]], y'[t] == x[t]Sin[x[t]y[t]]};
col = {Red, Blue, Black, Green};
bc = {{x[0] == -1, y[0] == 0.2}, {x[0] == -1.5, y[0] == 0.5},
      {x[0] == 0, y[0] == -0.2}, {x[0] == 1.5, y[0] == -0.2}};
pp = Table[
    se = Join[sys, bc[[i]]];
    u = NDSolve[se, {x, y}, {t, 0, 12}];
    p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, 0, 12},
        PlotStyle -> {Thickness[0.02], col[[i]]};
    p, {i, 1, 4}];
vp = StreamPlot[{yCos[xy], xSin[xy]}, {x, -5, 5}, {y, -5, 5}];
Show[vp, pp]

```



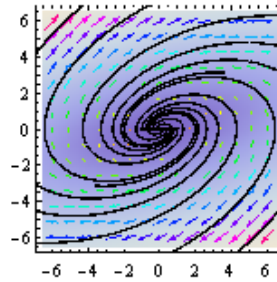
Пример. Большую наглядность дает функция `VectorDensityPlot`, которая кроме векторного поля строит плотность некоторого скалярного поля, например, поля длины/нормы векторного поля. Она раскрашивает область в цвета, яркость которых соответствует значению этого скалярного поля.

```

sys = {x'[t] == 2y[t], y'[t] == -x[t] + y[t]};
pp = Table[
    se = Join[sys, {x[0] == -1 + a, y[0] == 0}];
    u = DSolve[se, {x, y}, t];
    p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, 0, 2 $\pi$ };
    p, {a, 0, 2, 0.2}];

```

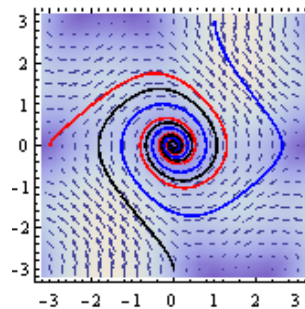
```
vp = VectorDensityPlot[{2y, -x + y}, {x, -6, 6}, {y, -6, 6},
    VectorColorFunction -> Hue];
Show[vp, pp, AspectRatio -> Automatic]
```



Пример. На фоне векторного поля и его нормы построим несколько линий тока этого векторного поля.

```
sys = {x'[t] == Sin[y[t]], y'[t] == -Sin[x[t]] - y[t]/4};
bc = {{x[0] == -3, y[0] == 0}, {x[0] == 1, y[0] == 3},
    {x[0] == 0, y[0] == -3}};

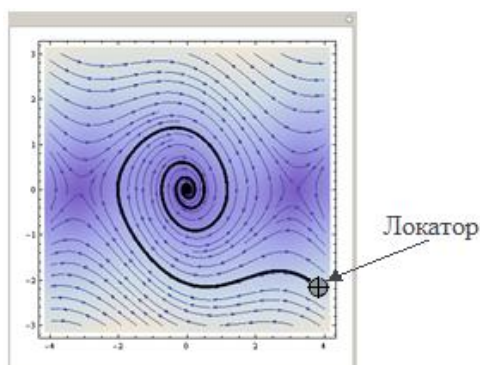
col = {Red, Blue, Black};
pp = Table[
    se = Join[sys, bc[[i]]];
    u = NDSolve[se, {x, y}, {t, 0, 30}];
    p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, 0, 30},
        PlotStyle -> {Thickness[0.01], col[[i]]}];
    p, {i, 1, 3}];
vp = VectorDensityPlot[{Sin[y], -Sin[x] - y/4}, {x, -3, 3}, {y, -3, 3},
    VectorPoints -> 21];
Show[vp, pp]
```



Пример. В манипуляторе можно использовать объект, который представляет «локатор» - точку, которую можно перемещать с помощью мыши. В нашем примере координаты «локатора» будут использоваться для задания начальных значений системы ОДУ, определяющей фазовую траекторию

```
Remove[x, y]; T = 100;
vp = StreamDensityPlot[{y, -Sin[x] - .25y}, {x, -4, 4}, {y, -3, 3}];
Manipulate[
    u = NDSolve[{x'[t] == y[t], y'[t] == -Sin[x[t]] - .25y[t],
        x[0] == pnt[[1]], y[0] == pnt[[2]]}, {x, y}, {t, 0, T}];
    p1 = ParametricPlot[Evaluate[{u[[1, 1, 2]][t], u[[1, 2, 2]][t]}], {t, 0, T},
        PlotRange -> {{-2, 2}, {-2, 2}}];

    Show[vp, p1],
    {{pnt, {1, 0}}, Locator}, SaveDefinitions -> True]
```



«Захватите» мышью «локатор» и перемещайте его. Каждое новое положение «локатора» задает новые начальные значения и, следовательно, новую линию тока (фазовую траекторию) системы. Кривая будет динамически перерисовываться.

Кстати, здесь мы использовали еще одну функцию `StreamDensityPlot`, изображающую линии тока на фоне графика плотности скалярного поля: в нашем случае на фоне поля нормы векторного поля.

Много других примеров представления решений ОДУ и их систем приведено в п. 4.5.

4.4 Дифференциально – алгебраические уравнения.

Система уравнений, связывающая неизвестные функции, может содержать алгебраические уравнения. Тогда ее называют системой дифференциально – алгебраических уравнений (система ДАУ).

```
ds = DSolve[{x'[t] + y'[t] == x[t] + t^2, x[t] - y[t] == 1}, {x[t], y[t]}, t];  
Simplify[ds[[1]]]
```

```
{x[t] → -8 - 4t - t^2 + 2et/2C[1], y[t] → -9 - 4t - t^2 + 2et/2C[1]}
```

Системы ДАУ могут содержать начальные условия

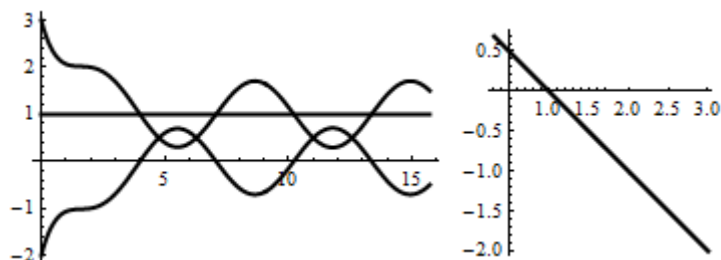
```
ds = DSolve[{x'[t] - y[t] == Sin[t], x[t] + y[t] == 1, x[0] == 3},  
{x[t], y[t]}, t];
```

```
Simplify[ds[[1]]]
```

```
Plot[Evaluate[{x[t], y[t], x[t] + y[t]}/. ds], {t, 0, 5π},  
PlotStyle → {{Black, Thickness[0.01]}}]
```

```
ParametricPlot[Evaluate[{x[t], y[t]}/. ds], {t, 0, 5π},  
PlotStyle → {{Black, Thickness[0.02]}}]
```

```
{x[t] →  $\frac{1}{2}(2 + 5e^{-t} - \cos[t] + \sin[t])$ , y[t] →  $\frac{1}{2}(-5e^{-t} + \cos[t] - \sin[t])$ }
```



Смысл решенной здесь задачи состоит в определении параметрического уравнения алгебраической кривой, уравнение которой использовалось в системе ДАУ. В нашем случае это было уравнение прямой $x + y = 1$. На правом графике мы построили эту прямую по найденным функциям $x(t)$, $y(t)$.

В системе ДАУ обязательно, чтобы хотя бы одно уравнение было дифференциальным. Вот пример системы двух ОДУ и одного алгебраического уравнения

DSolve[$\{x'[t] == x[t] + 2y[t], y'[t] == x[t] + z[t],$
 $x[t] + y[t] + z[t] == 0\}, \{x[t], y[t], z[t]\}, t]$

$\{\{x[t] \rightarrow e^t C[1] + \frac{26}{27} e^{-t} C[2] + e^{-t} (-1 + e^{2t}) C[2],$

$y[t] \rightarrow \frac{1}{27} e^{-t} C[2],$

$z[t] \rightarrow -e^t C[1] - e^{-t} C[2] - e^{-t} (-1 + e^{2t}) C[2]\}\}$

Вот пример системы одного ОДУ и двух алгебраических уравнений

DSolve[$\{x'[t] == x[t] + y[t], y[t] == x[t] - 2z[t], x[t] + 2z[t] == 0\},$
 $\{x[t], y[t], z[t]\}, t]$

$\{\{x[t] \rightarrow -\frac{1}{8} e^{3t} C[1], y[t] \rightarrow -\frac{1}{4} e^{3t} C[1], z[t] \rightarrow \frac{1}{16} e^{3t} C[1]\}\}$

При постановке задачи важно правильно задавать начальные условия. Например, рассмотрим задачу $x(t) + y'(t) = 0$, $y(t) = 0$, $x(0) = 1$, $y(0) = 0$. Имеем $y(t) = 0 \Rightarrow y'(t) = 0 \Rightarrow x(t) = 0$. Т.о. решением является функции $x(t) = 0$, $y(t) = 0$. Но это несовместимо с условием $x(0) = 1$.

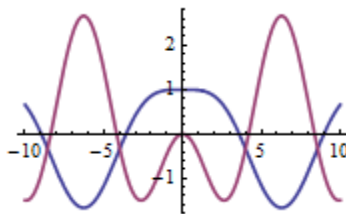
Для системы ДАУ, содержащей ДУ 2-го порядка, надо задавать не менее 2 – х начальных условий

eqs = $\{x''[t] == \frac{1}{4} y[t], x[t] + y[t] == \text{Cos}[t], x[0] == 1, x'[0] == 0\}$

s = **DSolve**[**eqs**, $\{x[t], y[t]\}, t]$

Plot[$\{x[t]/.s[[1]], y[t]/.s[[1]]\}, \{t, -10, 10\}, \text{PlotStyle} \rightarrow \text{Thickness}[0.01]$]

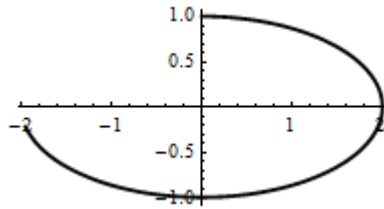
$\{\{x[t] \rightarrow \frac{1}{3} (4\text{Cos}[\frac{t}{2}] - \text{Cos}[t]), y[t] \rightarrow \frac{4}{3} (-\text{Cos}[\frac{t}{2}] + \text{Cos}[t])\}\}$



Символьное решение нелинейных систем ДАУ является трудной задачей и во многих случаях вместо **DSolve** можно использовать **NDSolve**.

s = **NDSolve**[$\{x'[t] == 2y[t] + x[t]y[t], \frac{x[t]^2}{4} + y[t]^2 == 1, x[0] ==$
 $0\}, \{x, y\}, \{t, 0, 10\}]$

ParametricPlot[**Evaluate**[$\{x[t], y[t]\}/.s$], $\{t, 0, 10\}, \text{PlotRange} \rightarrow \text{All}, \text{AspectRatio} \rightarrow \text{Automatic}]$



4.5 Примеры исследования ОДУ

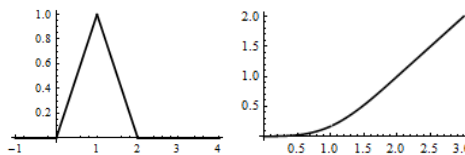
В этом параграфе мы приводим примеры решения прикладных задач из области физики, механики, электротехники и других областей науки, которые сводятся к решению ОДУ или их систем.

4.5.1 Движение материальной точки по прямой.

Дифференциальное уравнение движения имеет вид: $m\ddot{x} = f(t)$, где m – масса точки, $f(t)$ – внешняя сила.

Пример 1.1. Пусть внешняя сила на отрезке времени $[0, 1]$ задана в виде $f(t) = t$, на отрезке времени $[1, 2]$ равна $f(t) = 2 - t$, и равна 0 при $t > 2$. Тогда решить задачу в *Mathematica* можно так (полагаем $m = 1$).

```
f[t_] = Piecewise[{{0, t < 0}, {t, t < 1}, {2 - t, t < 2}}, 0];
Plot[f[t], {t, -1, 4}]
ds = DSolve[{x''[t] == f[t], x[0] == 0, x'[0] == 0}, x, t]
xs = x /. ds[[1]];
Plot[xs[t], {t, 0, 3}]
```



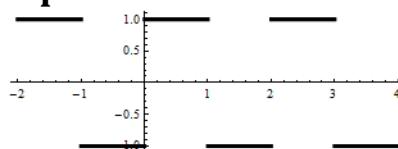
На левом рисунке приведен график внешней силы, а на правом – график функции $x(t)$ – решения ОДУ.

Пример 1.2. Внешняя сила является периодической импульсивной функцией, которая на разных участках периода принимает только два значения ± 1 . Определим такую функцию (*Periodic Impuls Function* = *Pif*)

$$Pif(x, a, w) = 2 \cdot \left(\left\lceil \frac{x}{w} \right\rceil - \left\lceil \frac{x-a}{w} \right\rceil \right) - 1$$

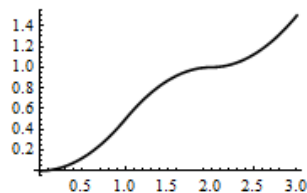
Это периодическая с периодом w функция. Для $a < w$ на отрезке $0 \leq x < a$ функция равна 1, на остальном куске периода $a \leq x < w$ функция равна -1 (случай $a \geq w$ мы не рассматриваем). На следующем рисунке приведен график функции $Pif(x, 1, 2)$.

```
Pif[x_, a_, w_] = 2(Floor[ $\frac{x}{w}$ ] - Floor[ $\frac{x-a}{w}$ ]) - 1;
Plot[Pif[x, 1, 2], {x, -2, 4}, AspectRatio -> Automatic]
```



Пусть внешняя сила имеет вид $f(t)=Pif(x,1,2)$ и $m=1$. Тогда

```
nds = NDSolve[{x''[t] == Pif[t, 1, 2], x[0] == 0, x'[0] == 0}, x, {t, 0, 4}]
xs = x /. nds[[1]];
Plot[xs[t], {t, 0, 3}]
```



4.5.2 Движение упругого мяча

Упругий мячик имеет начальное положение и скорость. Сопротивление воздуха пренебрежимо мало, но энергия движения расходуется при отскоке мяча от земли. Пусть при отскоке от земли вертикальная скорость мячика составляет 90% от вертикальной скорости в момент падения.

Уравнение движения (без учета сопротивления воздуха) имеет вид

$$m\ddot{\mathbf{r}} = -m\mathbf{g},$$

где \mathbf{g} — вектор ускорения свободного падения, имеющий направление вертикально вниз. В покомпонентной форме имеем $\ddot{x} = 0$, $\ddot{y} = -g$ и начальные условия $x(0) = x_0$, $y(0) = h$, $\dot{x}(0) = v_0^x$, $\dot{y}(0) = v_0^y$. Векторное уравнение распадается на два независимых скалярных уравнения.

$$\begin{cases} \ddot{x}(t) = 0 \\ \ddot{y}(t) = -g \end{cases},$$

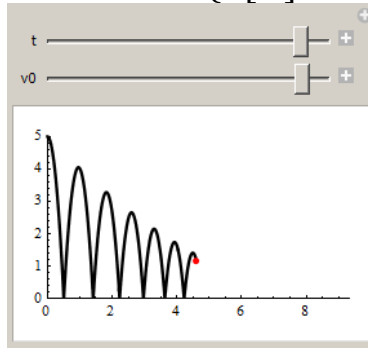
первое из которых имеет решение $x(t) = x_0 + v_0^x \cdot t$. Это указывает на то, что в горизонтальном направлении движение происходит с постоянной скоростью v_0^x . В нашем примере положим $x_0 = 0$. Второе уравнение также интегрируется, если не учитывать скачкообразного изменения скорости в момент касания земли. Но мы хотим показать, как можно использовать функцию `WithEvent`, и будем решать уравнение численно. В момент отскока t_k вертикальная скорость \dot{y} меняет знак, т.е. $\dot{y}_{\text{после отскока}}(t_k + \varepsilon) = -0.9 \cdot \dot{y}_{\text{до отскока}}(t_k - \varepsilon)$ и становится положительной.

Находим решение второго уравнения системы с учетом резкого изменения скорости мяча при отскоке. Затем строим параметрическое уравнение траектории в манипуляторе с параметрами времени движения и начальной скорости.

Manipulate[

```
s = NDSolveValue[{y''[t] == -9.81, y[0] == 5, y'[0] == 0,
WhenEvent[y[t] == 0, y'[t] -> -0.9 y'[t]], y, {t, 0, 10}];
p1 = ParametricPlot[Evaluate[{x[t], s[t]}], {t, 0, tmax},
PlotRange -> {{0, 10v0}, {0, 5}}];
p2 = Graphics[{RGBColor[1, 0, 0], Disk[{x[tmax], s[tmax]}, 0.1]}];
Show[p1, p2],
```

```
{{tmax, 0.5, t}, 0.3, 10}, {{v0, 0.5, "v0"}, 0.1, 1},
AutoAction → False, Initialization: > {x[t_]:= v0t; }
```



4.5.3 Равновесие струны под действием сосредоточенных сил.

Пусть струна закреплена в точках $x_0=0$ и $x_n=L$, а внешние сосредоточенные силы f_i приложены в точках x_i : $0=x_0 < x_1 < x_2 < \dots < x_{n-1} < x_n=L$. Уравнение

равновесия струны (с малым отклонением) имеет вид: $u''_{xx} = -f(x)$, $f(x) = \frac{F(x)}{T}$,

где T – натяжение струны, $F(x)$ – внешняя сила, рассчитанная на единицу длины, $u(x)$ – отклонение струны от положения равновесия. Функцию $f(x)$, создаваемую набором сосредоточенных сил во внутренних точках x_i отрезка

$[0, L]$, можно представить следующим образом $f(x) = \sum_{i=1}^{n-1} f_i \cdot \delta(x - x_i)$, где

$\delta(x - x_i)$ – функция Дирака, равная нулю везде, кроме точки x_i в которой она обращается в бесконечность так, что интеграл по отрезку, содержащему точку x_i , равен единице. В пакете *Mathematica* функция Дирака называется `DiracDelta[x]`. В примерах положим $T=1$.

Пример 3.1. Пусть $L=3$, $x_1=1$, $x_2=2$, $f_1=1$, $f_2=2$. Тогда

$L=3$;

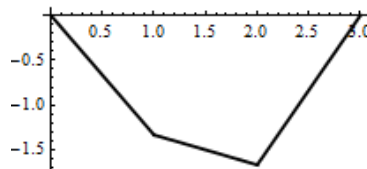
$f[x_] = \text{DiracDelta}[x - 1] + 2\text{DiracDelta}[x - 2]$;

$\text{ds} = \text{DSolve}[\{u''[x] == f[x], u[0] == 0, u[L] == 0\}, u, x]$

$z[x_] = \text{Simplify}[(\text{ds}[[1, 1, 2, 2]])]$

$\text{Plot}[z[x], \{x, 0, L\}]$

$-\frac{4x}{3} + 2(-2+x)\text{HeavisideTheta}[-2+x] + (-1+x)\text{HeavisideTheta}[-1+x]$



Мы изменили знак правой части специально, чтобы на графике прогиб струны был направлен вниз. Интересно отметить, что решение можно представить в

более «приятном» виде: $u(x) = -\frac{5}{2} + \frac{x}{6} + \frac{1}{2}|x-1| + |x-2|$, который *Mathematica*

не умеет получать с помощью функции `Simplify`. Для этого в полученном выражении для $z[x]$ нужно сделать замену

$(-k+x)\text{HeavisideTheta}[-k+x] \rightarrow 1/2(x-k-\text{Abs}[x-k])$,

где k надо положить 1 и 2. Эта замена следует из формулы $x H(x) = \frac{1}{2}(x + |x|)$, где $H(x)$ – функция Хэвисайда (HeavisideTheta). Такая замена в более общем виде нам потребуется несколько раз, а именно $x^k H(x) = \frac{1}{2}(x^k + |x|^k)$ при нечетном k и $x^k H(x) = \frac{1}{2}(x^k + x^{k-1} |x|)$ при четном k . Напишем функцию, выполняющую такую замену

```
SimplifyHeavisideTheta = Function[{expr, xL, k, x},
  n = Length[xL];
  ch = Table[HeavisideTheta[-xL[[i]] + x], {i, n}];
  If[Mod[k, 2] == 0,
    cuh = Table[(-xL[[i]] + x)^k HeavisideTheta[-xL[[i]] + x] →
      1/2((x - xL[[i]])^k + (x - xL[[i]])^{k-1} Abs[x - xL[[i]]]), {i, n}];
    cuh = Table[(-xL[[i]] + x)^k HeavisideTheta[-xL[[i]] + x] →
      1/2((x - xL[[i]])^k + Abs[x - xL[[i]]]^k), {i, n}];
  ];
  cu = Simplify[Collect[expr, ch];
  c2 = Simplify[(cu/.cuh)];
  cu2 = Simplify[Collect[c2, ch];
  Simplify[(cu2/.cuh)]
];
```

Эта функция упрощает выражение $expr$, содержащее элементы вида $(-i + x)^k HeavisideTheta[-i + x]$, xL – список целых значений i , входящих в заменяемые выражения, k – целая положительная степень $k=1,2,\dots$; x – имя переменной в выражении $expr$.

Тогда, используя эту функцию, имеем

```
SimplifyHeavisideTheta[z[x], {1, 2}, 1, x]//TraditionalForm
```

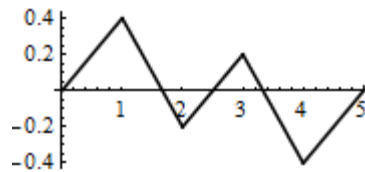
$$\frac{1}{6}(6|x-2| + 3|x-1| + x - 15)$$

Обратите внимание, что здесь решена краевая задача, а не задача Коши, как в предыдущем пункте.

Пример 3.2. Пусть $L=5$, сосредоточенные нагрузки $f_{list} = [1, -1, 1, -1]$ приложены в точках $x_i = 1, 2, 3, 4$.

```
L = 5; xl = {1, 2, 3, 4}; fl = {1, -1, 1, -1};
f[x_] = Sum[fl[[i]] DiracDelta[x - xl[[i]]], {i, Length[xl]}];
nds = DSolve[{u''[x] == -f[x], u[0] == 0, u[L] == 0}, u, x];
z[x_] = (nds[[1, 1, 2, 2]]);
SimplifyHeavisideTheta[z[x], xl, 1, x]//TraditionalForm
Plot[z[x], {x, 0, L}, AspectRatio → 0.5]
```

$$\frac{1}{10}(5|x-4| - 5|x-3| + 5|x-2| - 5|x-1| + 4x - 10)$$



Пример 3.3. Равновесие струны под действием кусочно – постоянной силы.

$L = 3$;

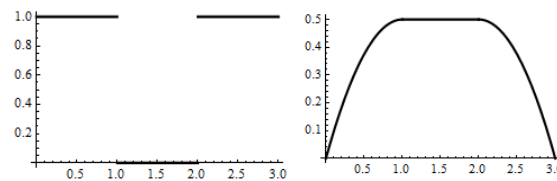
$f[x_] = \text{HeavisideTheta}[x] - \text{HeavisideTheta}[x - 1] +$
 $\text{HeavisideTheta}[x - 2];$

$\text{ds} = \text{DSolve}[\{u''[x] == -f[x], u[0] == 0, u[L] == 0\}, u, x];$

$\text{SimplifyHeavisideTheta}[\text{ds}[[1, 1, 2, 2]], \{0, 1, 2\}, 2, x] // \text{TraditionalForm}$

$\text{GraphicsRow}[\{\text{Plot}[f[x], \{x, 0, L\}], \text{Plot}[\text{nds}[[1, 1, 2, 2]], \{x, 0, L\}]\}$

$\frac{1}{4}(-x | x | - (x - 2) | x - 2 | + (x - 1) | x - 1 | - x^2 + 6x - 3)$



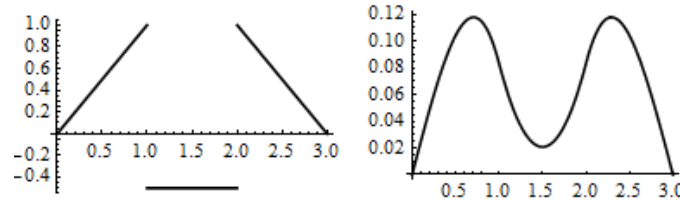
Слева показан график внешней силы, справа профиль струны.

Пример 3.4. Равновесие нити под действием кусочно - линейной внешней силы

$L = 3; f[x_] = \text{Piecewise}[\{\{x, x < 1\}, \{-1/2, x < 2\}\}, 3 - x];$

$\text{ds} = \text{DSolve}[\{u''[x] == -f[x], u[0] == 0, u[L] == 0\}, u, x];$

$\text{GraphicsRow}[\{\text{Plot}[f[x], \{x, 0, L\}], \text{Plot}[\text{ds}[[1, 1, 2, 2]], \{x, 0, L\}]\}$



Слева показан график внешней силы, справа профиль нити. Заметим, что когда исходная функция $f[x]$ задана как кусочная (Piecewise), то и результат получается в виде Piecewise функции. Если ту же функцию $f[x]$ задать с использованием функции Хэвисайда, то результат можно будет привести к выражению, содержащему функции абсолютного значения.

$f[x_] = x(1 - \text{HeavisideTheta}[x - 1]) - (\text{HeavisideTheta}[x - 1]$
 $- \text{HeavisideTheta}[x - 2])/2 + (3 - x)\text{HeavisideTheta}[x - 2];$

$\text{ds} = \text{DSolve}[\{u''[x] == -f[x], u[0] == 0, u[L] == 0\}, u, x];$

$\text{ss} = \text{SimplifyHeavisideTheta}[\text{ds}[[1, 1, 2, 2]], \{1, 2\}, 2, x]$

$\text{ss} // \text{TraditionalForm}$

$\frac{1}{24}((x - 2)(2x - 13)|x - 2| + (x - 1)(2x + 7)|x - 1| - 9(2(x - 3)x + 5))$

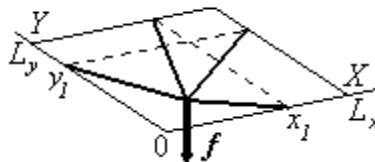
или

$\text{Collect}[\text{Expand}[\text{ss}], \{\text{Abs}[-1 + x], \text{Abs}[-2 + x]\}] // \text{TraditionalForm}$

$\left(\frac{x^2}{12} - \frac{17x}{24} + \frac{13}{12}\right)|x - 2| + \left(\frac{x^2}{12} + \frac{5x}{24} - \frac{7}{24}\right)|x - 1| - \frac{3x^2}{4} + \frac{9x}{4} - \frac{15}{8}$

4.5.4 Статический прогиб сети нитей

Пример 4.1 Рассмотрим механическую систему, состоящую из двух пересекающихся под прямым углом струн/нитей L_x и L_y длиной L . Нити в точке пересечения (x_1, y_1) соединены и в этой точке приложена сосредоточенная сила f_0 . Под действием силы f_0 обе нити примут вид ломаной (см. следующий рисунок)



Требуется составить уравнения этих ломаных.

Очевидно, что действие силы f_0 частично уравнивается струной L_x и частично струной L_y . Т.е. на каждую струну действует своя сосредоточенная сила. Это значит, что на нить L_x действует только часть силы f_0 . Обозначим ее через a . Часть силы f_0 , действующую на нить L_y , обозначим через b . Очевидно, что $a + b = f_0$. Если знать силы a и b , то можно, решив соответствующее ОДУ, определить прогиб каждой из струн.

Имея ввиду впоследствии рассмотреть сеть нитей, т.е. множество нитей с прогибами $u_1(x), \dots$ и $v_1(y), \dots$ обозначим функцию прогибов нити L_x через $u[1][x]$, а нити L_y – через $v[1][y]$, а параметры через $a[1]$ и $b[1]$. Составим и решим ОДУ прогиба первой и второй нитей при $x_1 = 1, y_1 = 2$

$L = 3; f_0 = 3;$

$dsX = DSolve[\{u[1]''[x] == -a[1]DiracDelta[x - 1],$
 $u[1][0] == 0, u[1][L] == 0\}, \{u[1][x]\}, x];$

$uu[x_] = SimplifyHeavisideTheta[dsX[[1, 1, 2]], \{1\}, 1, x]$

$dsY = DSolve[\{v[1]''[y] == -b[1]DiracDelta[y - 2],$
 $v[1][0] == 0, v[1][L] == 0\}, \{v[1][y]\}, y];$

$vv[y_] = SimplifyHeavisideTheta[dsY[[1, 1, 2]], \{2\}, 1, y]$

$\frac{1}{6}a[1](3 + x - 3Abs[-1 + x])$
 $-\frac{1}{6}(-6 + y + 3Abs[-2 + y])b[1]$

В решении присутствуют неопределенные пока параметры $a[1]$ и $b[1]$. Но у нас есть еще два условия: прогиб нитей в точке пересечения одинаков, и сумма этих параметров равна приложенной в узле (x_1, y_1) силе f_0 . Составляем алгебраическую систему уравнений и решаем ее

$ss = Solve[\{uu[1] == vv[2], a[1] + b[1] == f_0\}, \{a[1], b[1]\}]$

$\{\{a[1] \rightarrow \frac{3}{2}, b[1] \rightarrow \frac{3}{2}\}\}$

Зная параметры $a[1]$ и $b[1]$, построим функции прогибов нитей и нарисуем их графики

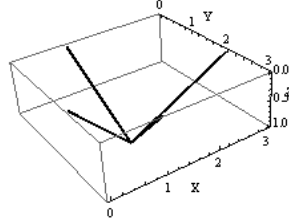
$U1[x_] = uu[x]/.ss[[1]]$

$V1[y_] = vv[y]/.ss[[1]]$

$$\frac{1}{4}(3+x-3\text{Abs}[-1+x])$$

$$\frac{1}{4}(6-y-3\text{Abs}[-2+y])$$

```
su1 = ParametricPlot3D[{t, 2, U1[t]}, {t, 0, L}];
sv1 = ParametricPlot3D[{1, t, V1[t]}, {t, 0, L}];
Show[su1, sv1, AxesLabel -> {"X", "Y", "Z"}]
```



Пример 4.2 Рассмотрим сеть, составленную из двух пар пересекающихся нитей/струн одинаковой длины и одинакового натяжения, имеющие симметричное расположение в плоскости гипотетической квадратной мембраны размера L . Пусть струны в точках пересечения соединены и в узлах приложены одинаковые вертикальные сосредоточенные силы f . В каждом узле (i, j) такая сила уравнивается частично натяжением нити одного семейства и частично натяжением нити второго семейства. Можно сказать, что на эти нити действуют по отдельности сосредоточенные силы, сумма которых равна внешней силе приложенной в этом узле. Пусть уравнение ломаных первого семейства будут $u_1(x)$ и $u_2(x)$, а уравнения ломаных второго семейства $v_1(y)$ и $v_2(y)$. Обозначим узлы сети через (i, j) , где $i, j=1,2$, сосредоточенные силы, приложенные к нитям первого семейства, через a_{ij} , и сосредоточенные силы, приложенные к нитям второго семейства – через b_{ij} . Очевидно, что $a_{ij}+b_{ij}=f$ для любых i и j . Рассматривая a_{ij} и b_{ij} как параметры, составим и решим соответствующие ОДУ прогиба нитей

```
Remove[a, b, u, v, UU, VV, U, V]
```

```
L = 3; XL = Table[i, {i, L - 1}]; YL = Table[j, {j, L - 1}];
```

```
m = Length[XL]; n = Length[YL];
```

```
f = Table[1, {i, m}, {j, n}];
```

```
dsX = DSolve[
```

```
  Flatten[Table[
    {u[j]''[x] == -Sum[a[j, k]DiracDelta[x - XL[[k]]], {k, m}],
    u[j][0] == 0, u[j][L] == 0}, {j, n}],
```

```
  Table[u[j][x], {j, n}], x];
```

```
sx = SimplifyHeavisideTheta[FullSimplify[dsX[[1]]], XL, 1, x];
```

```
Table[UU[j][x_] = sx[[j, 2]], {j, n}];
```

```
dsY = DSolve[Flatten[Table[
```

```
  {v[i]''[y] == -Sum[b[i, k]DiracDelta[y - YL[[k]]], {k, n}],
  v[i][0] == 0, v[i][L] == 0}, {i, m}],
```

```
  Table[v[i][y], {i, m}], y];
```

```
sy = SimplifyHeavisideTheta[FullSimplify[dsY[[1]]], YL, 1, y];
```

```
Table[VV[i][y_] = sy[[i, 2]], {i, m}];
```

Здесь функции $UU[j][x]$ и $VV[i][y]$ содержат неопределенные пока константы $a[j,k]$ и $b[i,k]$. Составим систему линейных алгебраических уравнений относительно этих величин и решим ее.

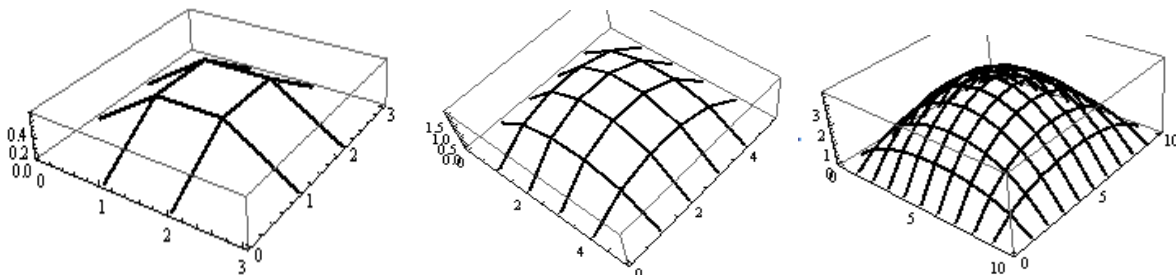
```
tt = Join[
  Flatten[Table[UU[j][XL[[i]]] == VV[i][YL[[j]]], {i, m}, {j, n}]],
  Flatten[Table[a[i, j] + b[j, i] == f[[i, j]], {i, m}, {j, n}]]];
tv = Flatten[Join[Table[a[i, j], {i, m}, {j, n}], Table[b[i, j], {i, m}, {j, n}]]];
ss = Solve[tt, tv];
```

```
{{a[1,1] -> 1/2, a[1,2] -> 1/2, a[2,1] -> 1/2, a[2,2] -> 1/2,
b[1,1] -> 1/2, b[1,2] -> 1/2, b[2,1] -> 1/2, b[2,2] -> 1/2}}
```

Зная параметры $a[j,k]$ и $b[i,k]$, выполним их подстановку в функции $UU[j][x]$ и $VV[i][y]$, и построим кривые прогибов нитей $U[j][x]$ и $V[i][y]$.

```
Table[U[j][x_] = UU[j][x]/.ss[[1]], {j, n}];
Table[V[i][y_] = VV[i][y]/.ss[[1]], {i, m}];
p = Table[ParametricPlot3D[{t, YL[[j]], U[j][t]}, {t, 0, L},
  DisplayFunction -> Identity], {j, n}];
q = Table[ParametricPlot3D[{XL[[i]], t, V[i][t]}, {t, 0, L},
  DisplayFunction -> Identity], {i, m}];
```

```
Show[p, q]
```



На предыдущем рисунке слева показана сеть, составленная из двух пар ортогонально пересекающихся упругих нитей. Если положить в предыдущем коде (сценарии) $L=5$, то получится сеть, показанная на предыдущем рисунке в середине. Положив $L=10$, мы получим деформированную сеть, показанную справа.

4.5.5 Прогиб балки

Уравнение изгибающего момента свободно опертой балки имеет тот же вид,

что и уравнение равновесия струны $\frac{d^2 M}{dx^2} = -q$, где $q(x)$ внешняя сила, а $M(x)$

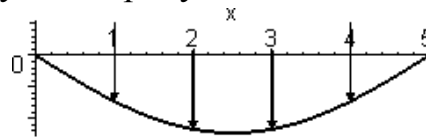
изгибающий момент в балке, x – координата, направленная вдоль оси балки. Это значит, если балка находится под действием сосредоточенных или кусочных внешних сил, то мы можем применить прежний код для определения изгибающего момента $M(x)$. Для определения прогиба балки нужно решать

уравнение $EJ \frac{d^2 u(x)}{dx^2} = -M(x)$, $J(x)$ – момент инерции поперечного сечения

балки, E – модуль упругости материала балки, $u(x)$ – отклонение нейтральной

оси балки от положения равновесия. Также можно сразу решать ОДУ относительно функции прогиба $\frac{d^2}{dx^2} \left(EJ \frac{d^2 u}{dx^2} \right) = q(x)$. Для свободно опертой на краях балки граничные условия должны иметь вид $u[0]=0, u[L]=0, u''[0]=0, u''[L]=0$. При ином закреплении граничные условия будут другими.

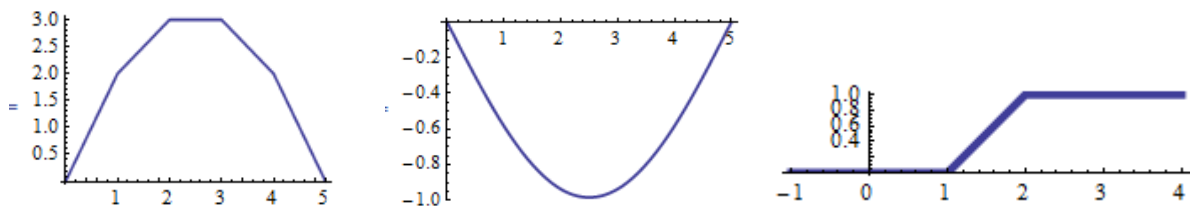
Пример 5.1 Пусть балка свободно оперта по краям и находится под действием четырех одинаковых сосредоточенных нагрузок. Нагрузки приложены на одинаковом расстоянии друг от друга и от концов балки. Решим модельную задачу, в которой положим $L=5$ (длина балки), $E=8$, $J=1$, $h=1$ (высота балки). Единичные сосредоточенные нагрузки приложены в точках $x_i = 1, 2, 3, 4$. Схема нагрузок показана на следующем рисунке.



Найдем изгибающий момент $M(x)$ балки. Его график показан на следующем рисунке слева.

```
Remove[M, f, u, x, U, W]; L = 5;
f[x_] = Sum[DiracDelta[x - i], {i, 4}];
sM = DSolve[{M''[x] == -f[x], M[0] == 0, M[L] == 0}, M, x];
M = sM[[1, 1, 2]];
SimplifyHeavisideTheta[M[x], {1, 2, 3, 4}, 1, x] // TraditionalForm
Plot[M[x], {x, 0, L}, PlotStyle -> Thickness[0.01]]
```

$$\frac{1}{2} (10 - |x-1| - |x-2| - |x-3| - |x-4|)$$



Находим форму нейтральной оси балки.

```
Eu = 8; J = 1;
sU = DSolve[{Eu * J * u''[x] == M[x], u[0] == 0, u[L] == 0}, u, x];
W = sU[[1, 1, 2]];
ss = SimplifyHeavisideTheta[W[x], {1, 2, 3, 4}, 3, x];
U[x_] = Simplify[Expand[ss]]
U[x] // TraditionalForm
Plot[U[x], {x, 0, L}, PlotStyle -> Thickness[0.01]]
```

$$\frac{1}{96} (100 - 150x + 30x^2 - |x-1|^3 - |x-2|^3 - |x-3|^3 - |x-4|^3)$$

График функции $U(x)$ показан на предыдущем рисунке в середине.

Целью следующего блока кода является графическое представление продольных напряжений в балке. Вначале напишем код, который рисует прямоугольную балку. Он использует вспомогательную функцию

$P[x, a, w] = (w + \text{Abs}[x - a] - \text{Abs}[x - a - w])/2;$

$\text{Plot}[P[x, 1, 1], \{x, -1, 4\}, \text{AspectRatio} \rightarrow \text{Automatic}]$

График функции $P(x, a, w)$ при $a=1, w=1$ показан на предыдущем рисунке справа.

Теперь составляем параметрическое уравнение области прямоугольника, представляющего балку (с методикой составления параметрических уравнений ограниченных областей вы можете познакомиться по пособиям автора, представленным в разделах спецкурса «Аналитические методы геометрического моделирования»). Для графического представления области балки мы используем параметрическое уравнение ее поверхности в пространстве с третьей координатой $Z=0$.

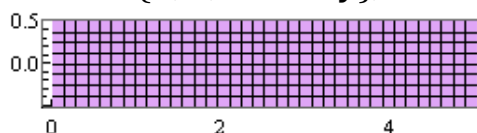
$h = 1; hh = h/2; (* \text{полувысота балки} *)$

$x[u, v_] = P[u, 0, L];$

$y[u, v_] = -hh + P[v, -hh, 2hh];$

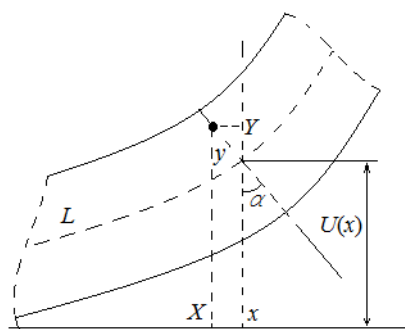
$\text{ParametricPlot3D}[\{x[u, v], y[u, v], 0\}, \{u, 0, L\}, \{v, -hh, hh\},$

$\text{ViewPoint} \rightarrow \{0, 0, \text{Infinity}\}, \text{Mesh} \rightarrow \{35, 7\}]$



Используя уравнения плоской области $x(u, v)$, $y(u, v)$ недеформированной балки и уравнение $U(x)$ ее нейтральной оси, можно написать параметрическое уравнение области деформированной балки.

В соответствии с гипотезой плоских сечений, все поперечные сечения, которые были перпендикулярны нейтральной оси балки до деформирования, остаются перпендикулярными нейтральной оси балки после ее деформирования.



Точки, имевшие до деформирования координаты (x, y) , переходят в точки с координатами (X, Y) , где $X = x - y \cos \alpha$, $Y = U(x) + y \cos \alpha$ (см. рисунок). Но

$\text{tg } \alpha = U'(x)$ и, поэтому, $X = x - \frac{U'_x(x)y}{\sqrt{1 + (U'_x(x))^2}}$ и $Y = U(x) + \frac{y}{\sqrt{1 + (U'_x(x))^2}}$. Тогда

функции $X(u, v)$, $Y(u, v)$, представляющие параметрические уравнения области изогнутой балки будут иметь вид

$$X(u, v) = x(u, v) - \frac{U'_x(x(u, v))y(u, v)}{\sqrt{1 + (U'_x(x(u, v)))^2}}$$

$$Y(u, v) = U(x(u, v)) + \frac{y(u, v)}{\sqrt{1 + (U'_x(x(u, v)))^2}}$$

Строим область деформированной балки. Для этого вычисляем производную $U_1(x) = U'_x(x)$ и создаем вспомогательную функцию $\sqrt{1 + (U'_x(x))^2}$.

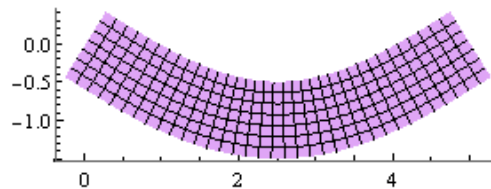
U1[x_] = Simplify[U'[x]/. Table[Abs[-i + x]^2 Abs'[-i + x] → (-i + x) Abs[-i + x], {i, 4}], x ∈ Reals]

Usq[x_] = $\sqrt{1 + U1[x]^2}$;

X[u_, v_] = $x[u, v] - \frac{U1[x[u, v]]y[u, v]}{Usq[x[u, v]]}$;

Y[u_, v_] = $U[x[u, v]] + \frac{y[u, v]}{Usq[x[u, v]]}$;

ParametricPlot3D[{X[u, v], Y[u, v], 0}, {u, 0, L}, {v, -hh, hh}, ViewPoint → {0, 0, Infinity}, Mesh → {34, 6}]



Следующим шагом будет показать в цвете продольные напряжения σ_x в балке. Если поперечное сечение балки прямоугольное, то σ_x вычисляются по формуле

$\sigma_x(x, y) = \frac{M(x) \cdot y}{J}$, где $M(x)$ – момент инерции в сечении x балки, y –

вертикальное смещение точки балки относительно нейтральной оси. Чтобы использовать эту функцию в качестве функции цвета, ее нужно привести к диапазону $[0, 1]$. Для этого нужно знать минимальные и максимальные значения напряжения σ_x . В нашей задаче они, очевидно, равны напряжениям в срединном сечении балки в верхней и нижней точках

$\sigma_{x \max/\min} = \pm \frac{M(L/2) \cdot (h/2)}{J}$, где h – высота балки.

Создаем функцию продольных напряжений **tensionX** и нормированную в диапазоне от 0 до 1 функцию **beamColor**.

tensionX[u_, v_] = $M[x[u, v]] \cdot y[u, v]/J$;

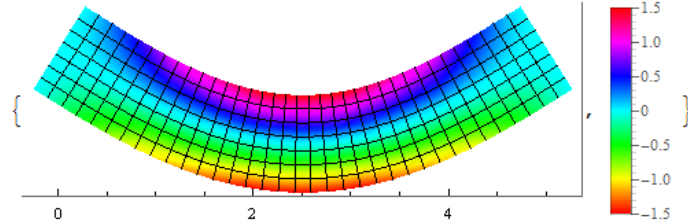
tensmin = $-\frac{M[L/2]hh}{J}$;

tensmax = $\frac{M[L/2]hh}{J}$;

beamColor[u_, v_] = $\frac{\text{tensionX}[u, v] - \text{tensmin}}{\text{tensmax} - \text{tensmin}}$;

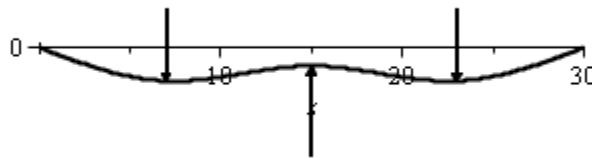
Рисуем область изогнутой балки, раскрашенную в соответствии с продольными напряжениями. Рядом рисуем палитру напряжений.

```
{ParametricPlot3D[{X[u, v], Y[u, v], 0}, {u, 0, L}, {v, -hh, hh},
  ViewPoint → {0, 0, Infinity}, Mesh → {34, 6},
  ColorFunction → (Hue[beamColor[#4, #5]] &),
  ColorFunctionScaling → False, Axes → {True, False, False}],
BarLegend[{Hue, {-1.5, 1.5}}, LegendMarkerSize → 150]}
```



□

Пример 5.2 Пусть балка свободно оперта по краям и находится под действием трех сосредоточенных нагрузок. Схема приложения нагрузок показана на следующем рисунке.



Решим задачу, в которой положим $L=30$, $E=2$, $J=54$, $h=6$. Сосредоточенные нагрузки $f_i = 4, -5, 4$ приложены в точках $x_i = 7, 15, 23$. Приведем только строки кода, которые изменены по сравнению с предыдущей задачей

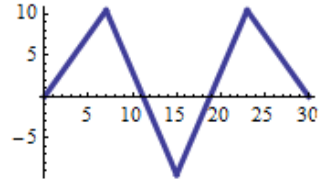
```
Remove[M, f, u, x, U, W]; L = 30;
```

```
f[x_] = 4DiracDelta[x - 7] - 5DiracDelta[x - 15] + 4DiracDelta[x - 23]
```

```
...
```

```
SimplifyHeavisideTheta[M[x], {7, 15, 23}, 1, x]//TraditionalForm
```

```
...
```



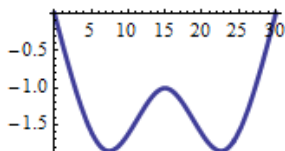
$$M[x] = \frac{1}{2}(-4|x - 23| + 5|x - 15| - 4|x - 7| + 45)$$

```
Eu = 2; J = 54;
```

```
...
```

```
ss = SimplifyHeavisideTheta[W[x], {7, 15, 23}, 3, x];
```

```
...
```



$$\frac{-4|x - 23|^3 + 5|x - 15|^3 - 4|x - 7|^3 + 135x^2 - 4050x + 33165}{1296}$$

```
h = 6; hh = h/2;
```

```
...
```

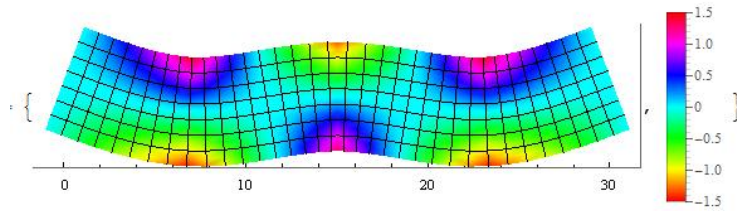
```
U1[x_] = Simplify[U'[x]/.{Abs[-7 + x]^2 Abs'[-7 + x] → (-7 + x) Abs[-7 + x],
  Abs[-15 + x]^2 Abs'[-15 + x] → (-15 + x) Abs[-15 + x],
  Abs[-23 + x]^2 Abs'[-23 + x] → (-23 + x) Abs[-23 + x]}, x ∈ Reals]
```

```
...
```

$$\text{tensmin} = -\frac{M[7]hh}{J};$$

$$\text{tensmax} = \frac{M[7]hh}{J};$$

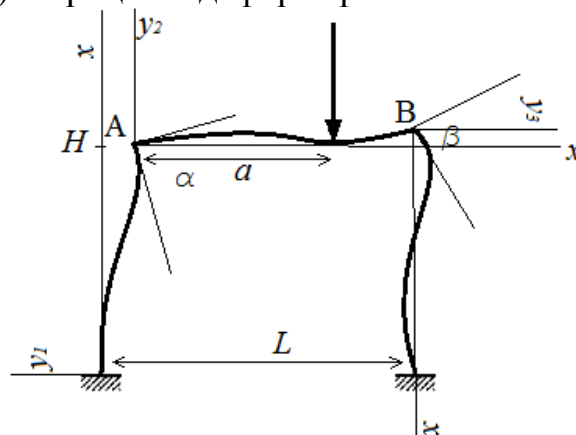
...



Для экономии места мы также не привели промежуточные рисунки, которые генерирует наш код.

4.5.6 Расчет рамы

Жесткой рамой называется такая стержневая система у которой все узловые соединения являются жесткими. Жесткий узел характеризуется тем, что угол между осями стержней его образующих не изменяется при действии нагрузки. Такой пример показан на следующем рисунке – углы α и β между касательными к осям горизонтальной и вертикальным балкам остается неизменным (прямым) в процессе деформирования.

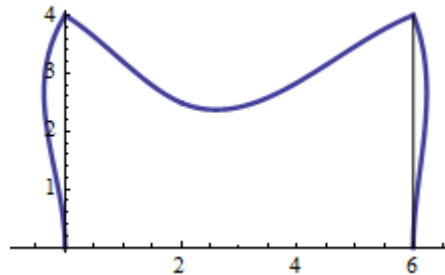


Когда рама деформируется, то обе касательные, проведенные в узле А, поворачиваются на одинаковый угол, иначе величина угла α изменилась бы. Тоже самое можно сказать и о касательных в точках В. С математической точки зрения расчет рамы сводится к решению системы ОДУ, описывающих деформацию стержней, задания граничных условий закрепления и сопряжения.

Рассмотрим раму, показанную на предыдущем рисунке. Размеры таковы: $L=6$, $H=4$, $a=2$. Положим также, что вертикальные и горизонтальные смещения узлов А и В равны нулю (они малы и ими можно пренебречь). Свяжем с каждым стержнем свою систему координат (x, y_i) ($i=1,2,3$), направления осей которых показаны на рисунке. Прогиб левого вертикального стержня обозначим через $y_1(x)$ ($0 \leq x \leq H$), прогиб верхнего горизонтального обозначим через $y_2(x)$ ($0 \leq x \leq L$), прогиб правого вертикального стержня – через $y_3(x)$ ($0 \leq x \leq H$). Условия неизменности углов (жесткие узлы А и В) будут выполнены, если в этих точках совпадают производные к соответствующим

прогибам. Т.е. эти условия имеют вид $y_1'(H) = y_2'(0)$, $y_2'(L) = y_3'(0)$. Также в этих узлах одинаковые изгибающие моменты сопрягаемых стержней $E_1 J_1 \frac{d^2 y_1(H)}{dx^2} = E_2 J_2 \frac{d^2 y_2(0)}{dx^2}$ и $E_2 J_2 \frac{d^2 y_2(L)}{dx^2} = E_3 J_3 \frac{d^2 y_3(0)}{dx^2}$. Условия жесткого защемления опор дают граничные условия вида $y_1(0) = 0$, $y_1'(0) = 0$, $y_3(H) = 0$, $y_3'(H) = 0$. Тогда для решения задачи имеем следующий код

```
H = 4; L = 6; E1 = 1; J1 = 1; E2 = 1; J2 = 1; E3 = 1; J3 = 1;
ss = DSolve[{
  E1J1u1''''[x] == 0,
  E2J2u2''''[x] == -DiracDelta[x - 2],
  E3J3u3''''[x] == 0,
  u1[0] == 0, u1[H] == 0, u1'[0] == 0, E1J1u1''[H] == E2J2u2''[0],
  u1'[H] == u2'[0], u2[0] == 0, u2[L] == 0, u3[0] == 0,
  E2J2u2''[L] == E3J3u3''[0], u2'[L] == u3'[0], u3[H] == 0,
  u3'[H] == 0}, {u1, u2, u3}, x]
y1 = u1/.ss[[1]];
y2 = u2/.ss[[1]];
y3 = u3/.ss[[1]];
p1 = ParametricPlot[{-y1[t], t}, {t, 0, H}];
p2 = ParametricPlot[{t, H + y2[t]}, {t, 0, L}];
p3 = ParametricPlot[{L + y3[t], H - t}, {t, 0, H}];
p4 = Graphics[Line[{{L, 0}, {L, H}}]];
Show[{p1, p2, p3, p4}, PlotRange -> {{-1, L + 1}, {-1, H + 1}}]
```



Для представления формы деформированной рамы, мы записали параметрические уравнения кривых(осей стержней) в глобальной системе координат. Выражения для прогибов в локальных координатах имеют вид

```
y1[x]
SimplifyHeavisideTheta[y2[x], {2}, 3, x] //TraditionalForm
y3[x]
-11/288 (-4x^2 + x^3)
1/324 (-27 |x - 2|^3 + 11x^3 + 63x^2 - 522x + 216)
7/288 (16x - 8x^2 + x^3)
```

4.5.7 Поперечные колебания струны с грузами

Пример 7.1 *Поперечные колебания струны с двумя грузами.* Имеется 2 груза массы M , расположенные на струне длиной $L=3a$ в точках $x=a$ и $x=2a$. Отрезки струны между грузами одинаковы, невесомы и подчиняются закону Гука. Натяжение в равновесии равно T . Обозначим вертикальное отклонение грузов от положения равновесия на оси струны через $y_1(t)$ и $y_2(t)$. Ограничимся рассмотрением малых колебаний. Уравнения движения грузов имеют вид

$$M \frac{d^2 y_1}{dt^2} = T \left(\frac{y_2 - y_1}{a} \right) - T \left(\frac{y_1 - y_0}{a} \right)$$

$$M \frac{d^2 y_2}{dt^2} = T \left(\frac{y_3 - y_2}{a} \right) - T \left(\frac{y_2 - y_1}{a} \right)$$

где y_0, y_3 - смещения левого и правого концов закрепленной струны, т.е. нули. Положим $a=1, M=1, T=1$. Тогда получим систему

$$\begin{cases} \frac{d^2 y_1}{dt^2} = y_2 - 2y_1 \\ \frac{d^2 y_2}{dt^2} = -2y_2 + y_1 \end{cases}$$

Рассмотрим колебание системы без начальной скорости. Для этого решим систему ОДУ относительно функций $y_1(t), y_2(t)$ с единичными начальными смещениями грузов и нулевыми начальными скоростями.

Remove[y1,y2]

sys = {y1''[t] == y2[t] - 2y1[t], y2''[t] == -2y2[t] + y1[t],
y1[0] == 1, y2[0] == 1, y1'[0] == 0, y2'[0] == 0}

ds = DSolve[sys, {y1, y2}, t]

y1 = ds[[1, 1, 2]]

y2 = ds[[1, 2, 2]]

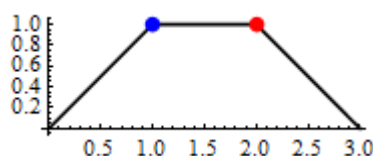
Зная функции $y_1(t), y_2(t)$, составим кусочное параметрическое уравнение струны и нарисуем начальное состояние системы

x[τ_, t_] = τ;

y[τ_, t_] = Piecewise[{{y1[t]τ, τ < 1},
{y1[t] + (y2[t] - y1[t])(τ - 1), τ < 2},
{y2[t] - y2[t](τ - 2), τ < 3}}, 0]

ParametricPlot[{x[t, 0], y[t, 0]}, {t, 0, 3},

pilog-> {PointSize[Large], Blue, Point[{1, y1[0]}], Red, Point[{2, y2[0]}]}]

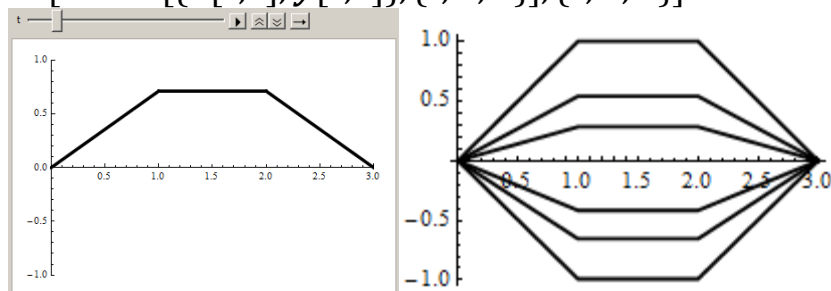


Теперь построим анимацию колебаний системы (след. рисунок слева) и график нескольких положений струны в различные моменты времени (рисунок справа).


```

Animate[
  ParametricPlot[Evaluate[{x[τ, t], y[τ, t]}], {τ, 0, 3},
    PlotRange → {{0, 3}, {-1, 1}},
    {t, 0, 6}, AnimationRunning → False]
ParametricPlot[Table[{x[t, i], y[t, i]}, {i, 0, 5}], {t, 0, 3}]

```

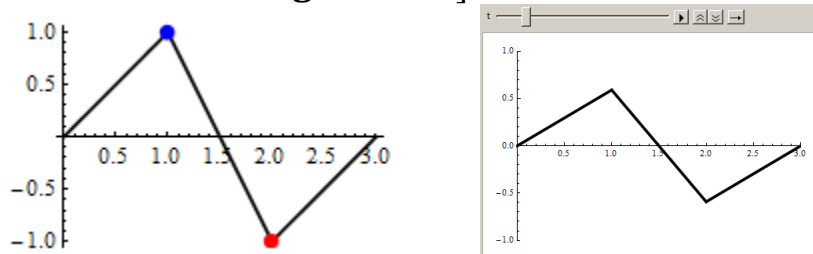


Полученное синхронное колебание обеих масс представляет первую моду колебаний системы. Для построения второй моды колебаний, представляющей движение масс в «противофазе», можно повторить весь предыдущий код, только надо задать начальные смещения тоже в «противофазе».

```

Remove[y1, y2]
sys = {y1''[t] == y2[t] - 2y1[t], y2''[t] == -2y2[t] + y1[t],
  y1[0] == 1, y2[0] == -1, y1'[0] == 0, y2'[0] == 0}
ds = DSolve[sys, {y1, y2}, t]
y1 = ds[[1, 1, 2]]
y2 = ds[[1, 2, 2]]
x[τ_, t_] = τ;
y[τ_, t_] = Piecewise[{{y1[t]τ, τ < 1},
  {y1[t] + (y2[t] - y1[t])(τ - 1), τ < 2},
  {y2[t] - y2[t](τ - 2), τ < 3}}, 0]
ParametricPlot[{x[t, 0], y[t, 0]}, {t, 0, 3},
  pilog -> {PointSize[Large], Blue, Point[{1, y1[0]}], Red, Point[{2, y2[0]}]}]
Animate[
  ParametricPlot[Evaluate[{x[τ, t], y[τ, t]}], {τ, 0, 3},
    PlotRange → {{0, 3}, {-1, 1}},
    {t, 0, 6}, AnimationRunning → False]

```



Начальное состояние системы показано на предыдущем рисунке слева, а панель анимации — справа. Для создания произвольных колебаний (суперпозиции первой и второй мод) достаточно задать произвольные начальные смещения. Попробуйте, например, задать условия $y_1(0)=1$, $y_2(0)=0$.

Пример 7.2 Поперечные колебания струны с n грузами. Имеется n грузов массы M , расположенных в точках $x = a, 2a, 3a, \dots, na$. Отрезки невесомой струны/нити между грузами одинаковы и подчиняются закону Гука. Натяжение в равновесии равно T . Обозначим вертикальное отклонение грузов от положения равновесия на оси струны через $y_i(t)$ ($i = 1, 2, \dots, n$). Ограничимся рассмотрением малых колебаний. Уравнения движения имеют вид

$$M \frac{d^2 y_i}{dt^2} = T \left(\frac{y_{i+1} - y_i}{a} \right) - T \left(\frac{y_i - y_{i-1}}{a} \right) \quad (i = 1, 2, \dots, n)$$

При этом, в силу условия закрепления струны на концах, следует считать, что $y_0(t) = 0$, $y_{n+1}(t) = 0$. Рассмотрим колебание системы без начальной скорости.

Положим $a=1$, $M=1$, $T=1$. Тогда имеем

$$\frac{d^2 y_i}{dt^2} = y_{i+1} - 2y_i + y_{i-1} \quad (i = 1, 2, \dots, n)$$

Напишем код для решения этой системы ОДУ и построим анимацию движения грузов.

$n = 32$;

**eqns = Flatten[Table[{ $y_i''[t] == (y_{i+1}[t] - 2y_i[t] + y_{i-1}[t])$,
 $y_i[0] == \text{Sin}[\frac{\pi i}{n+1}] + \text{Sin}[\frac{2\pi i}{n+1}]$, $y_i'[0] == 0$ }, { i, n }]]];**

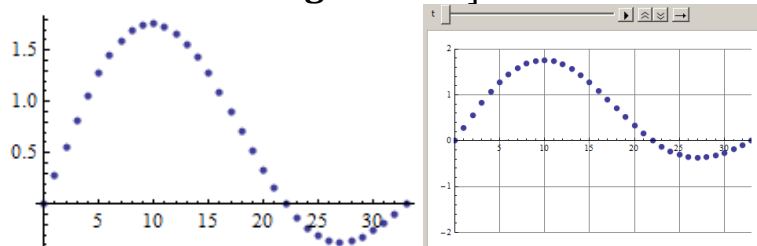
eqns = Join[eqns, { $y_0[t] == 0$, $y_{n+1}[t] == 0$ }];

vars = Table[y_i , { $i, 0, n + 1$ }];

sol = NDSolve[eqns, vars, { $t, 0, 2(n + 1)$ }];

ListPlot[Table[{ $i, y_i[0]$ } /. sol[[1]], { $i, 0, n + 1$ }], PlotStyle → PointSize[0.02]]

**Animate[
ListPlot[Table[{ $i, y_i[t]$ } /. sol[[1]], { $i, 0, n + 1$ }],
PlotStyle → PointSize[0.02], PlotRange → {{0, $n + 1$ }, {-2, 2}},
GridLines → Automatic],
{ $t, 0, 2n$ }, AnimationRunning → False]**

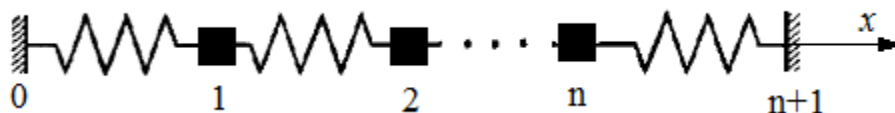


Здесь на левом графике показано начальное состояние системы, справа – панель анимации.

Попробуйте использовать большее/меньшее n . Задайте другие начальные смещения, например, $y_i[0] == \frac{n}{2} - \text{Abs}[i - \frac{n}{2}]$.

4.5.8 Модель Ферми – Улама – Пасты

Представим простейшую одномерную модель кристалла – цепочку одинаковых грузиков массы m , соединенных одинаковыми пружинками жесткостью k . Обозначим через y_i отклонение (вдоль оси X) грузика i от положения равновесия.



Тогда, в соответствии с законом Гука, уравнения движения i -го грузика имеет вид

$$m \frac{d^2 y_i}{dt^2} = k(y_{i+1} - y_i) - k(y_i - y_{i-1}) \quad (i = 1, 2, \dots, n),$$

который полностью совпадает с уравнениям поперечных колебаний невесомой струны с грузами.

Ферми с коллегами С. Уламом и Дж. Пастой предположил, что сила $F(y_{i+1}, y_i)$, с которой взаимодействуют $(i+1)$ -й и i -й атомы имеет небольшую нелинейную добавку. Это значит, что возвращающая сила имеет вид $F(y_{i+1}, y_i) = k \Delta l + \alpha \Delta l^2$, где $\Delta l = y_{i+1} - y_i$ и α мало. Это приводит к следующей системе ОДУ

$$m \frac{d^2 y_i}{dt^2} = k(y_{i+1} - 2y_i + y_{i-1}) + \alpha((y_{i+1} - y_i)^2 - (y_i - y_{i-1})^2) \quad (i = 1, 2, \dots, n)$$

При решении этой системы Ферми с коллегами следили не за отдельными частицами, а за поведением мод колебаний, которые при $\alpha = 0$ совпадают с синусоидами.

Рассмотрим движение, при котором в начальный момент времени возбуждена только одна первая мода с периодом T . Код решения этой задачи повторяет код предыдущего примера с той разницей, что уравнения системы ОДУ теперь нелинейные. Положим $k=1$, $n=32$, $\alpha=1$ и начальные смещения зададим в форме первой моды $y_i(0) = \sin\left(\frac{\pi i}{n+1}\right)$.

$n = 32; \alpha = 1;$

$T = 66;$ (* период линейных колебаний *)

eqns1 = Flatten[Table[

$\{y_i''[t] == (y_{i+1}[t] - 2y_i[t] + y_{i-1}[t]) +$
 $\alpha \cdot ((y_{i+1}[t] - y_i[t])^2 - (y_i[t] - y_{i-1}[t])^2),$

$y_i[0] == \text{Sin}[\frac{\pi i}{n+1}], y_i'[0] == 0\}, \{i, n\}];$

eqns1 = Join[eqns1, $\{y_0[t] == 0, y_{n+1}[t] == 0\};$

vars1 = Table[$y_i, \{i, 0, n+1\}$];

sol1 = NDSolve[eqns1, vars1, $\{t, 0, 200(n+1)\}$, MaxSteps $\rightarrow 100000$];

p1 = Table[ListPlot[Table[

$\{i, y_i[\text{tk}]\} /. \text{sol1}[[1]], \{i, 0, n+1\},$

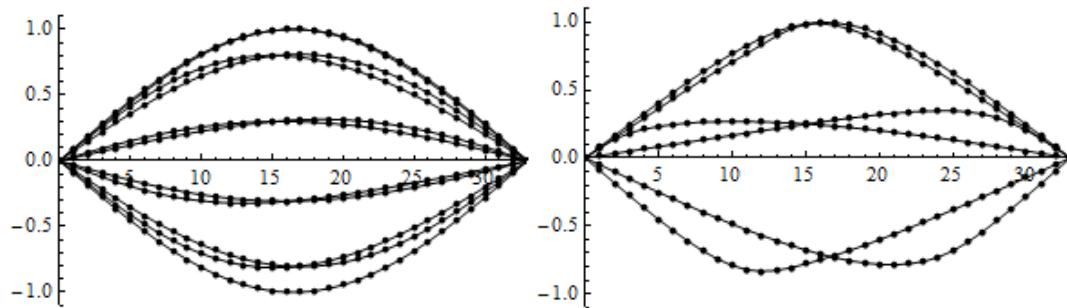
PlotStyle $\rightarrow \{\text{PointSize}[0.01], \text{Red}\}$, PlotRange $\rightarrow \text{All}$,

Joined $\rightarrow \text{True}$, Mesh $\rightarrow \text{All}$],

$\{\text{tk}, 0, T, T/10\}$];

Show[p1, PlotRange $\rightarrow \{\{0, n+1\}, \{-1.1, 1.1\}\}$]

Обратите внимание, что при решении системы ОДУ использована опция `MaxSteps→100000`, которая необходима для большого временного интервала, на котором определяется численное решение.



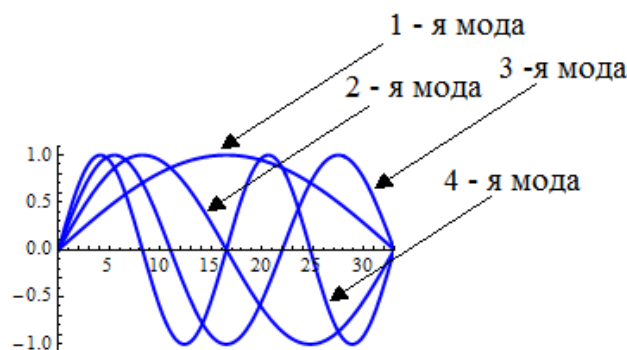
На предыдущем рисунке слева показаны формы колебаний в моменты времени $t = 0, \frac{T}{10}, \frac{2T}{10}, \dots, T$. Следующий код строит графики решения при $t = 2T, 2T + \frac{T}{5}, \dots, 3T$, которые показаны на предыдущем рисунке справа. Для наглядности на обоих графиках смещение грузов отложены в вертикальном направлении.

```
p2 = Table[ListPlot[Table[
    {i, yi[tk]}/.sol1[[1]], {i, 0, n + 1}],
    PlotStyle → {PointSize[0.01], Red}, PlotRange → All,
    Joined → True, Mesh → All],
    {tk, 2T, 3T, T/5}];
Show[p2, PlotRange → {{0, n + 1}, {-1.1, 1.1}}]
```

Как видим, форма решения все еще похожа на половину волны синусоиды, т.е. на первую моду колебаний. Однако, уже наблюдается сдвиг максимума в стороны от середины отрезка $[0, n+1]$. Первая мода еще узнаваема при $t \approx 7T$.

Следующий код строит графики 4 - х первых мод линейных колебаний.

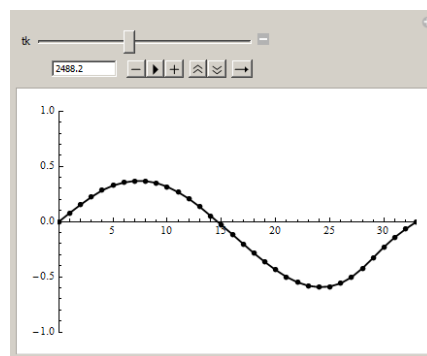
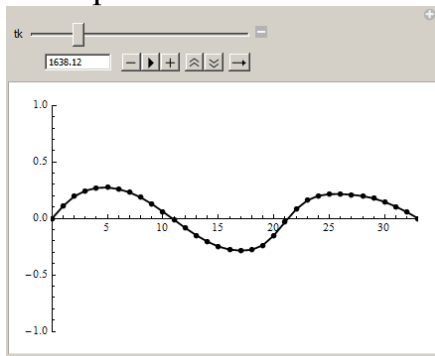
```
pl = Table[Plot[Sin[ $\frac{\pi x k}{n + 1}$ ], {x, 0, n + 1},
    PlotStyle → {Blue, Thickness[0.01]}], {k, 1, 4}];
Show[pl, PlotRange → {{0, n + 1}, {-1.1, 1.1}}]
```



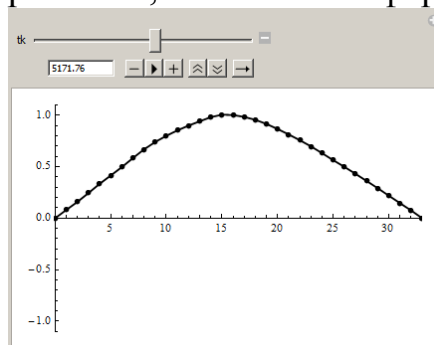
Вернемся к нелинейным колебаниям. При $t \approx 25T$ мы наблюдаем в основном 3 - ю моду. Чтобы в этом убедиться, достаточно использовать манипулятор (следующий рисунок слева) и код, приведенный ниже.

Manipulate[
ListPlot[Table[{ i , $y_i[tk]$ }/.sol1[[1]], { i , 0, $n + 1$ },
PlotStyle → {PointSize[0.015], Black, Thickness[0.005]},
PlotRange → {{0, $n + 1$ }, {-1, 1}}, Joined → True, Mesh → All],
{tk, 24T, 28T, 0.01T}]

При $t \approx 38.5T$ мы наблюдаем в основном 2 – ю моду. Измените в предыдущем коде последнюю строку на {tk, 36T, 40T, 0.01T}] и вы получите манипулятор, показанный справа.



Графики решений, построенные при $t \approx 51.5T$ снова напоминают 3 – ю моду. При $t \approx 77T$ мы снова возвращаемся к первой моде. Внесите изменение в последнюю строку предыдущего кода {tk, 75T, 80T, 0.01T}] и вы получите манипулятор с графиками решений, близкими по форме к 1 – й моде.



Подведем итог, который впервые наблюдали Ферми и его сотрудники. В начальные моменты времени форма колебаний близка к форме первой моды, но потом начинается перекачивание энергии в другие моды, и форма решения больше напоминает сумму нескольких синусоид с различным периодом (сумма мод). При $t \approx 25T$ возбуждена в основном 3 – я мода, при $t \approx 38.5T$ преобладает 2 – я мода. Затем при $t \approx 51.5T$ форма опять близка к 3 – й моде, и при $t \approx 77T$ снова возвращается к первой моде.

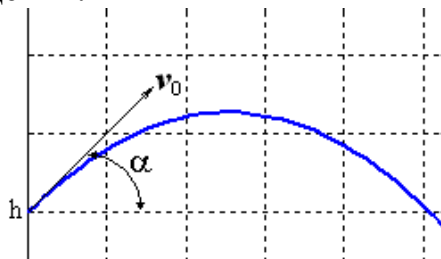
Численные эксперименты сотрудников Ферми показали, что такое поведение не случайность. Увеличение числа грузовиков, изменение значения параметра α , изменение формы нелинейности (например, $\alpha \Delta l^3$ вместо $\alpha \Delta l^2$) не меняет поведение системы. Моды не сливаются, а происходит выделение некоторых из них, которые доминируют по очереди. Когда возвращается начальная мода, все повторяется. Время возвращения T_R (в нашем примере $T_R \approx 77T$) зависит от n , от вида нелинейности, но «солирование» низших мод и возвращение при $t = T_R$ наблюдалось сотрудниками Ферми во всех расчетах.

4.5.9 Движение тела, брошенного под углом к горизонту

Решим задачу Коши, описывающую движение тела, брошенного с начальной скоростью v_0 под углом α к горизонту в предположении, что сопротивление воздуха пропорционально квадрату скорости. В векторной форме уравнение движения имеет вид

$$m\ddot{\mathbf{r}} = -\gamma \cdot \mathbf{v} |\mathbf{v}| - m\mathbf{g},$$

где $\mathbf{r}(t)$ радиус – вектор движущегося тела, $\mathbf{v} = \dot{\mathbf{r}}(t)$ – вектор скорости тела, γ – коэффициент сопротивления, $m\mathbf{g}$ – вектор силы веса тела массы m , \mathbf{g} – вектор ускорения свободного падения.



Особенность этой задачи состоит в том, что движение заканчивается в заранее неизвестный момент времени, когда тело падает на землю.

Если обозначить $k = \gamma / m$, то в координатной форме мы имеем систему уравнений

$$\ddot{x} = -k \dot{x} \sqrt{\dot{x}^2 + \dot{y}^2}$$

$$\ddot{y} = -k \dot{y} \sqrt{\dot{x}^2 + \dot{y}^2} - g$$

к которой следует добавить начальные условия: $x(0) = 0$, $y(0) = h$ (h начальная высота), $\dot{x}(0) = v_0 \cos \alpha$, $\dot{y}(0) = v_0 \sin \alpha$.

$$\alpha = \frac{\pi}{4}; v_0 = 1; h = 0; k = 0.01; g = 9.81;$$

```
sys = {x''[t] == -k x'[t] Sqrt[x'[t]^2 + y'[t]^2],
      y''[t] == -k y'[t] Sqrt[x'[t]^2 + y'[t]^2] - g,
      x[0] == 0, y[0] == h, x'[0] == v0 Cos[alpha], y'[0] == v0 Sin[alpha];
```

```
s = NDSolveValue[Join[sys,
  {WhenEvent[y[t] == 0, "StopIntegration"]}], {x, y}, {t, 0, Infinity}]
```

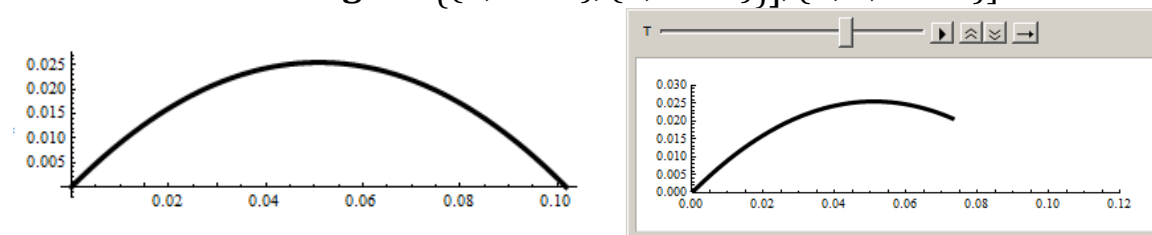
```
tmax = s[[1, 1, 1, 2]]
```

```
ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, tmax}] (рис. слева)
```

```
Animate[
```

```
ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, T},
```

```
PlotRange -> {{0, 0.12}, {0, 0.03}}, {T, 0, tmax}]
```

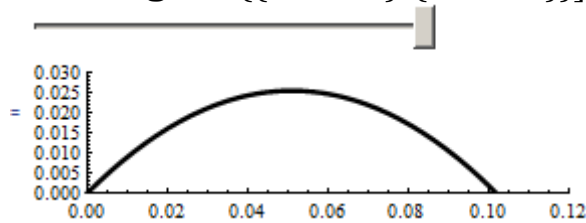


Чтобы для определения длительности полета нам не приходилось подбирать значение t_{\max} «на глазок», мы использовали функцию `WhenEvent` для

определения момента наступления события «высота полета равна нулю» и задали реакцию на это событие `StopIntegration` (остановка вычислений). Момент времени, в который произошло событие остановки вычислений, является параметром решения «`InterpolationFunction`» и доступ к нему мы получили с помощью индексации.

Вместо анимации можно использовать элемент «слайдер»

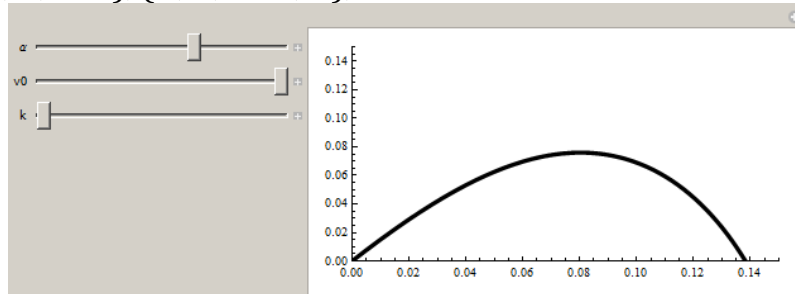
```
DynamicModule[{T = .01},
  {Slider[Dynamic[T], {T, 0.2}],
  Dynamic[
    ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]], {t, 0, T},
      PlotRange → {{0, 0.12}, {0, 0.03}}]]//TableForm]
```



Для наблюдением за поведением траектории тела при различных углах «бросания» α , начальных скоростях v_0 и коэффициентах сопротивления среды k удобно использовать манипулятор.

$h = 0; g = 9.81;$

```
Manipulate[
  sys = {x''[t] == -kx'[t] $\sqrt{x'$ [t]2 + y'[t]2,
    y''[t] == -ky'[t] $\sqrt{x'$ [t]2 + y'[t]2 - g, x[0] == 0, y[0] == h};
  s = NDSolveValue[Join[sys, {x'[0] == v0Cos[ $\alpha$ ], y'[0] == v0Sin[ $\alpha$ ]},
    {WhenEvent[y[t] == 0, "StopIntegration"]}], {x, y}, {t, 0,  $\infty$ ]];
  tmax = s[[1, 1, 1, 2]];
  ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]], {t, 0, tmax},
    PlotRange → {{0, 0.15}, {0, 0.15}}, AspectRatio → 0.5],
  {{ $\alpha$ ,  $\frac{\pi}{4}$ }, 0.1,  $\frac{\pi}{2}$  - 0.06, 0.1},
  {{v0, 1}, 0.1, 2, 0.1}, {k, 5, 100, 1}, ControlPlacement → Left]
```



4.5.10 Движение тел под действием тяготения

Пример 10.1 Исследуем задачу о движении планеты вокруг Солнца под действием тяготения.



Закон всемирного тяготения Ньютона в векторной форме имеет вид $m\mathbf{a} = F\mathbf{e}$, где \mathbf{a} – вектор ускорения, \mathbf{e} – единичный вектор, направленный от планеты к солнцу, $F = \gamma \frac{mM}{R^2}$ – величина силы притяжения, R – расстояние между притягивающимися телами, m – масса планеты, M – масса солнца, γ – гравитационная постоянная. Свяжем начало координат с массивным телом (солнцем). В любой момент времени функции $(x(t), y(t))$ представляют радиус-вектор планеты. Сила, с которой солнце притягивает планету, будет направлена вдоль единичного вектора

$$\mathbf{e} = \left(\frac{-x(t)}{\sqrt{x^2(t) + y^2(t)}}, \frac{-y(t)}{\sqrt{x^2(t) + y^2(t)}} \right)$$

имеющего противоположное направление радиус-вектору планеты. Тогда

$$m \begin{Bmatrix} x''(t) \\ y''(t) \end{Bmatrix} = -\gamma \frac{mM}{(x^2(t) + y^2(t))^{3/2}} \begin{Bmatrix} x(t) \\ y(t) \end{Bmatrix},$$

Введем обозначение $\gamma M = k$. Получим следующую систему обыкновенных дифференциальных уравнений относительно неизвестных функций $x(t)$, $y(t)$ – координат движущейся планеты:

$$\begin{cases} x''(t) = -\frac{k x(t)}{(x^2(t) + y^2(t))^{3/2}} \\ y''(t) = -\frac{k y(t)}{(x^2(t) + y^2(t))^{3/2}} \end{cases}$$

Для модельной задачи выберем $k=1$ и зададим начальные условия $x(0)=1$, $x'(0)=0$, $y(0)=0$, $y'(0)=1$. Решим задачу при погрешностях вычислений, заданных для функции `NDSolve` по-умолчанию, и при пониженных погрешностях.

$$\text{sys} = \left\{ \begin{aligned} x''[t] &== -\frac{x[t]}{(x[t]^2 + y[t]^2)^{3/2}}, y''[t] == -\frac{y[t]}{(x[t]^2 + y[t]^2)^{3/2}}, \\ x[0] &== 1, x'[0] == 0, y[0] == 0, y'[0] == 0.4 \end{aligned} \right\}$$

`s = NDSolveValue[sys, {x, y}, {t, 0, 20}]`

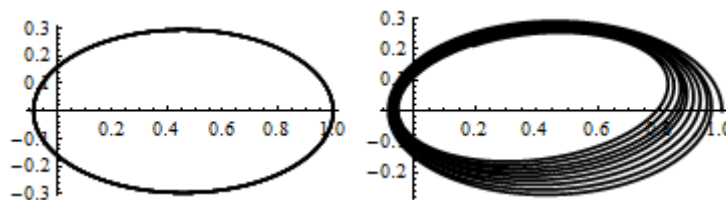
`sb = NDSolveValue[sys, {x, y}, {t, 0, 20},`

`AccuracyGoal → 3, PrecisionGoal → 6]`

`ParametricPlot[s[[1]][t], s[[2]][t], {t, 0, 20}, AspectRatio → Automatic]`

`ParametricPlot[sb[[1]][t], sb[[2]][t], {t, 0, 20}, AspectRatio → Automatic]`

На следующем рисунке показана получаемая траектория движения планеты (кривая с параметрическим уравнением $x(t), y(t)$)



Левая траектория получена при стандартной точности вычислений, правая – при пониженной. В решении `sb` отчетливо видно как накопление погрешности вычислений влияет на результат.

Можно построить анимацию движения

```
s = NDSolveValue[sys, {x, y}, {t, 0, 20}]
```

```
Animate[
```

```
  p1 = ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}, {t, 0, tmax},
```

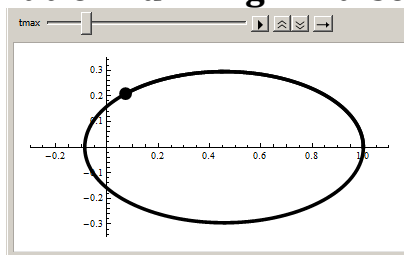
```
    PlotRange → {{-0.3, 1.1}, {-0.35, 0.35}}, PlotPoints → 1000];
```

```
  p2 = Graphics[Disk[{s[[1]][tmax], s[[2]][tmax]}, 0.025]];

```

```
  Show[{p1, p2}],
```

```
  {tmax, 0, 20, 0.1}, AnimationRunning → False]
```



Пример 10.2 Смоделируем задачу о запуске искусственного спутника на орбиту Луны. В этом случае в уравнениях движения спутника под действием тяготения будет присутствовать внешняя сила $\mathbf{f}(t)$, действующая в течение ограниченного времени. Ее величину и длительность действия надо подобрать так, чтобы спутник вышел на орбиту Луны.

Отметим, что спутник должен подняться на достаточную высоту, чтобы его орбита не захватывала поверхность планеты. Поэтому внешняя сила должна вначале быть вертикальной (или иметь большую вертикальную составляющую), а затем быть горизонтальной, чтобы придать спутнику необходимую орбитальную скорость. Если вертикальная составляющая ускорения будет не достаточной, то траектория спутника (эллипс) будет задевать поверхность планеты.

В качестве центрального тела будем рассматривать Землю, с которой свяжем начало координат. Луна движется под действием притяжения Земли и ее координаты обозначим $x_m(t), y_m(t)$. Координаты спутника обозначим $x_s(t), y_s(t)$, начальные значения которых должно быть на поверхности Земли. Притяжением Луны к спутнику можно пренебречь. Уравнения движения спутника будут иметь вид

$$\begin{cases} x_s''(t) = -\frac{\gamma M x_s(t)}{(x_s^2(t) + y_s^2(t))^{3/2}} + a_x(t) + \frac{\gamma m (x_s(t) - x_m(t))}{((x_s(t) - x_m(t))^2 + (y_s(t) - y_m(t))^2)^{3/2}} \\ y_s''(t) = -\frac{\gamma M y_s(t)}{(x_s^2(t) + y_s^2(t))^{3/2}} + a_y(t) + \frac{\gamma m (y_s(t) - y_m(t))}{((x_s(t) - x_m(t))^2 + (y_s(t) - y_m(t))^2)^{3/2}} \end{cases}$$

Здесь M – масса Земли, m – масса Луны, $(a_x(t), a_y(t))$ – вектор ускорения, вызываемый внешней силой (работающим двигателем ракеты). Заметим, что мы не учитываем факт изменения массы спутника, возникающего при сгорании топлива ракеты.

Луна также движется под действием притяжения Земли и уравнения ее движения (такие же как в предыдущем примере) имеют вид

$$\begin{cases} x_m''(t) = -\frac{\gamma M x_m(t)}{(x_m^2(t) + y_m^2(t))^{3/2}} \\ y_m''(t) = -\frac{\gamma M y_m(t)}{(x_m^2(t) + y_m^2(t))^{3/2}} \end{cases}$$

В результате мы получаем систему четырех ОДУ, к которым надо добавить начальные условия. Здесь мы рассмотрим модельную задачу с вымышленными значениями масс и размеров.

Начальные условия спутника определяют, что в нулевой момент времени ракета-носитель спутника находится на поверхности планеты (радиуса 1) и имеет небольшую нормальную к поверхности скорость (вдоль оси X)

$$x_s(0) = 1, \quad y_s(0) = 0, \quad x_s'(0) = 0.2, \quad y_s'(0) = 0.$$

Начальные условия для Луны определяют ее положение и скорость

$$x_m(0) = 1.9, \quad y_m(0) = 0, \quad x_m'(0) = 0, \quad y_m'(0) = 0.8$$

Решим задачу и построим анимацию движения Луны и спутника, а также анимацию видимого движения спутника с поверхности Луны.

Remove[xs,ys,xm,ym,x,y,x1,y1]

M = 10; m = 1; γ = 0.1; t0 = 30;

ax[t_] = Piecewise[{{0, t ≤ 0.2}, {1.4, t ≤ 1}, {-0.5, t ≤ 2}, {-0.9, t ≤ 2.5}, {1, t ≤ 3.1}, {0, t > 3.1}}];

ay[t_] = Piecewise[{{0, t ≤ 0.2}, {0, t ≤ 1}, {1.2, t ≤ 2}, {-0.2, t ≤ 2.5}, {-2, t ≤ 3.1}, {0, t > 3.1}}];

**eq1 = xs''[t] ==
$$\frac{-\gamma M xs[t]}{(xs[t]^2 + ys[t]^2)^{3/2}} + ax[t] + \frac{\gamma m (xm[t] - xs[t])}{((xm[t] - xs[t])^2 + (ym[t] - ys[t])^2)^{3/2}};$$**

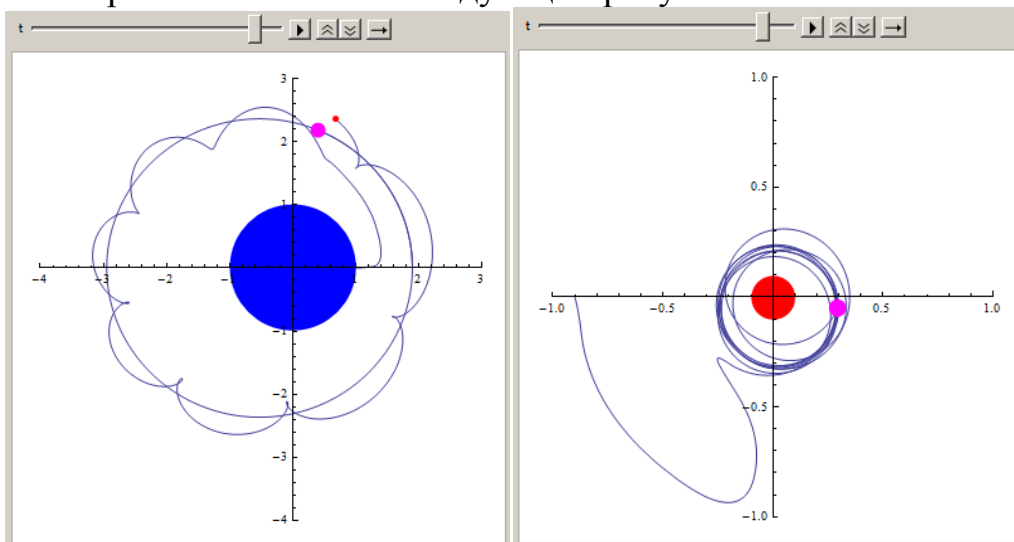
**eq2 = ys''[t] ==
$$\frac{-\gamma M ys[t]}{(xs[t]^2 + ys[t]^2)^{3/2}} + ay[t] + \frac{\gamma m (ym[t] - ys[t])}{((xm[t] - xs[t])^2 + (ym[t] - ys[t])^2)^{3/2}};$$**

```

eq3 = xm''[t] ==  $\frac{-\gamma M x_m[t]}{(x_m[t]^2 + y_m[t]^2)^{3/2}}$ ;
eq4 = ym''[t] ==  $\frac{-\gamma M y_m[t]}{(x_m[t]^2 + y_m[t]^2)^{3/2}}$ ;
bc = {xs[0] == 1, ys[0] == 0, xs'[0] == 0.2, ys'[0] == 0,
      xm[0] == 1.9, ym[0] == 0, xm'[0] == 0, ym'[0] == 0.8};
sys = Join[{eq1, eq2, eq3, eq4}, bc];
s = NDSolve[sys, {xs, ys, xm, ym}, {t, 0, t0}]
x = xs /. s[[1, 1]]; y = ys /. s[[1, 2]];
x1 = xm /. s[[1, 3]]; y1 = ym /. s[[1, 4]];
Animate[
  pp = ParametricPlot[{x[t], y[t]}, {t, 0, t}, PlotRange -> {{-4, 3}, {-4, 3}}];
  ss = ParametricPlot[{x1[t], y1[t]}, {t, 0, t}, PlotRange -> {{-4, 3}, {-4, 3}}];
  ssd = Graphics[{RGBColor[1, 0, 0], Disk[{x[t], y[t]}, 0.05]}];
  gr = Graphics[{RGBColor[1, 0, 1], Disk[{x1[t], y1[t]}, 0.12]}];
  earth = Graphics[{RGBColor[0, 0, 1], Disk[{0, 0}, 1]}];
  Show[{pp, ssd, ss, gr, earth}, Axes -> True,
    {t, 0.1, 30, 0.05}, AnimationRunning -> False]

```

Результат построения показан на следующем рисунке слева.



Следующий код строит траекторию движения спутника, наблюдаемую с Луны.

```

Animate[
  pp = ParametricPlot[{x[t] - x1[t], y[t] - y1[t]}, {t, 0, t},
    PlotRange -> {{-1, 1}, {-1, 1}}];
  ssd = Graphics[{RGBColor[1, 0, 0], Disk[{0, 0}, 0.1]}];
  gr = Graphics[{RGBColor[1, 0, 1], Disk[{x[t] - x1[t], y[t] - y1[t]}, 0.04]}];
  Show[{pp, ssd, gr}, Axes -> True,
    {t, 0.1, 30, 0.05}, AnimationRunning -> False]

```

Траектория спутника, наблюдаемая с Луны, показана на предыдущем рисунке справа.

4.5.11 Математический маятник

Исследуем поведения математического маятника. Пусть масса груза равна единице, а стержень, на котором подвешена масса, невесом. Тогда дифференциальное уравнение движения груза имеет вид

$$\varphi'' + k \varphi' + \omega^2 \sin \varphi = 0$$

где $\varphi(t)$ угол отклонения маятника от положения равновесия (нижнее положение), параметр k характеризует величину трения, $\omega^2 = g/l$ (g ускорение свободного падения, l – длина маятника). Для определения конкретного движения к уравнению движения надо добавить начальные условия $\varphi(0) = \varphi_0$, $\varphi'(0) = \varphi'_0$. Выберем следующие значения параметров $k=0.5$, $\omega^2=10$ и начальные значения $\varphi_0 = 0$, $\varphi'_0 = 5$.

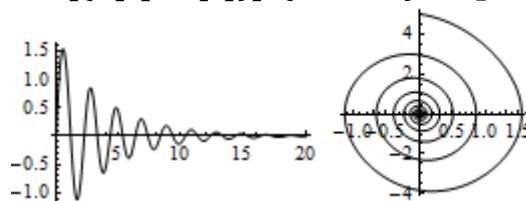
Решаем задачу, строим график решения (следующий рисунок слева) и фазовую траекторию (рисунок справа)

$k = 0.5$; $w = 10$;

**$s = \text{NDSolveValue}[\{x''[t] + kx'[t] + w\text{Sin}[x[t]] == 0, x[0] == 0,$
 $x'[0] == 5\}, x, \{t, 0, 20\}]$**

$\text{Plot}[s[t], \{t, 0, 20\}]$

$\text{ParametricPlot}[\text{Evaluate}[\{s[t], s'[t]\}], \{t, 0, 20\}, \text{AspectRatio} \rightarrow 1]$



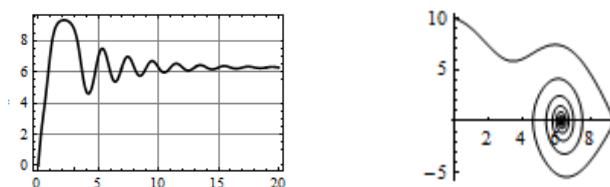
Как видно из левого графика, максимальный угол отклонения маятника не превышают $\pi/2$ и колебания маятника затухают.

Увеличим начальную скорость до 10. Решаем задачу, строим график решения (следующий рисунок слева) и фазовую траекторию (рисунок справа)

**$s = \text{NDSolveValue}[\{x''[t] + kx'[t] + w\text{Sin}[x[t]] == 0, x[0] == 0,$
 $x'[0] == 10\}, x, \{t, 0, 20\}]$**

$\text{Plot}[s[t], \{t, 0, 20\}]$

$\text{p1} = \text{ParametricPlot}[\text{Evaluate}[\{s[t], s'[t]\}], \{t, 0, 20\}, \text{AspectRatio} \rightarrow 1]$

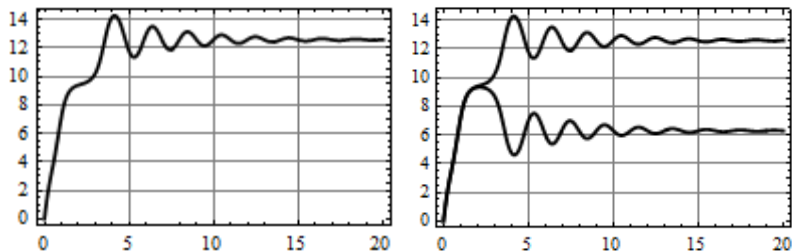


Максимальное значение угла составляет примерно 10 радиан. Маятник сделал один полный оборот вокруг точки закрепления (угол отклонения увеличился на $2\cdot\pi$), а затем колебания затухают в окрестности значения $2\cdot\pi$ (для маятника угол поворота $2\cdot\pi$ представляет то же, что и 0 радиан, т.е. положение равновесия).

Обратим внимание на следующий факт – точность вычислений в последнем примере имеет существенное значение. Снизим относительную и

абсолютную погрешности опциями `AccuracyGoal→3, PrecisionGoal→3` и найдем решение той же задачи

```
sbad = NDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0,  
  x[0] == 0, x'[0] == 10}, x, {t, 0, 20}, AccuracyGoal → 3, PrecisionGoal →  
3]  
p2 = Plot[sbad[t], {t, 0, 20}, GridLines → Automatic,  
  Frame → True, PlotRange → All]  
Show[p1, p2]
```



На правом рисунке показаны оба графика решений: второй более высокий, полученный при пониженной точности вычислений, и первый – полученный при стандартной точности. Отличие весьма значительное. Верхняя кривая представляет маятник, сделавший два полных оборота вокруг точки подвеса, а нижняя – маятник, сделавший только один полный оборот. Изменение точности привело к «качественному» изменению решения – два оборота маятника вместо одного!

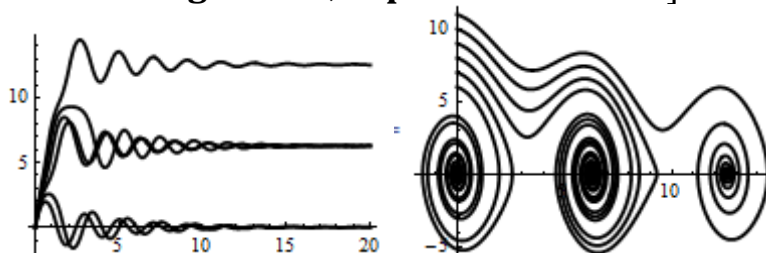
Построим несколько графиков угла отклонения (след. рис. слева) и фазовых траекторий (след. рис. справа), задавая различную начальную скорость.

$k = 0.5; w = 10;$

```
s = ParametricNDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0,  
  x[0] == 0, x'[0] == v0}, x, {t, 0, 20}, {v0}]
```

```
Plot[Table[s[v0][t], {v0, 6, 11, 1}], {t, 0, 20},  
  PlotStyle → {Thickness[0.01], {Black}}]
```

```
ParametricPlot[Table[{s[v0][t], s[v0]'[t]}, {v0, 6, 11, 1}], {t, 0, 20},  
  PlotRange → All, AspectRatio → 0.7]
```



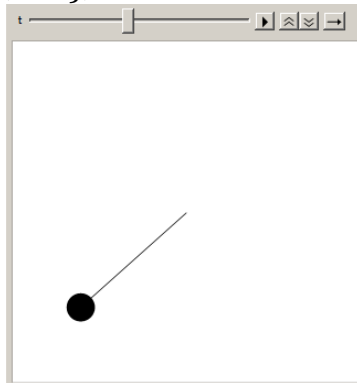
Как видим, начальная скорость при $v=6, 7$ недостаточна, чтобы маятник прошел верхнюю точку и сделал хотя бы один полный оборот. При начальной скорости $v=8, 9, 10$ маятник совершает один полный оборот, а затем его колебания затухают. При $v=11$ маятник смог выполнить два полных оборота и только после этого его колебания стали затухать вокруг положения равновесия.

Для наглядности, построим еще анимацию движения модели маятника

```

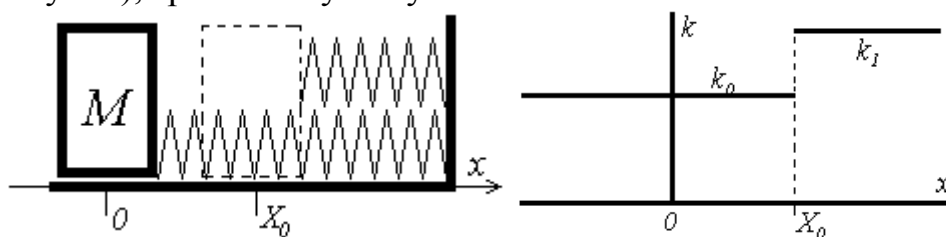
 $\varphi = \text{NDSolveValue}[\{x''[t] + kx'[t] + w\text{Sin}[x[t]] == 0,$ 
 $x[0] == 0, x'[0] == 10\}, x, \{t, 0, 20\}]$ 
g1[t_] = Line[{{0, 0}, {-Sin[ $\varphi[t]$ ], -Cos[ $\varphi[t]$ ]}];
d1[t_] = Disk[{-Sin[ $\varphi[t]$ ], -Cos[ $\varphi[t]$ ]}, 0.1];
Animate[Graphics[{g1[t], d1[t]}, PlotRange  $\rightarrow \{-1.1, 1.1\}, \{-1.1, 1.1\}$ ],
{t, 0, 4 $\pi$ }, AnimationRunning  $\rightarrow$  False]

```



4.5.12 Колебание массивного тела под действием пружин

Пример 12.1 Решим задачу о колебании массивного тела массы M на невесомых пружинах, одна из которых короче другой и короткая не привязана к телу (см. рисунок), трение отсутствует.



Для тела M выполняется уравнение движения $\ddot{x}(t) + k^2 x(t) = 0$. Здесь коэффициент k отвечает за жесткость пружины. При движении вправо и достижении положения X_0 тело M присоединяется ко второй пружине и жесткость системы меняется. Когда тело движется влево, и проходит положение X_0 , оно отрывается от второй пружины и жесткость уменьшается. Т.о. жесткость зависит от смещения груза M относительно положения равновесия, т.е. $k = k(x)$. При этом функция $k(x)$ является кусочно постоянной. График функции $k(x)$ приведен на предыдущем рисунке справа.

Пусть $k_0=1$, $k_1=1.5$ и начальные значения $x(0)=-1$, $x'(0)=v_0=1$. Для сравнения решим задачу при постоянном $k=1$ и при кусочно – постоянной жесткости k . Тогда

```

uc = NDSolve[{x''[t] + x[t] == 0, x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
up = NDSolve[{x''[t] + (1 + 0.5UnitStep[x[t]])^2 x[t] == 0,
x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
Plot[{x[t]/. uc[[1, 1]], x[t]/. up[[1, 1]]}, {t, 0, 20}]
ParametricPlot[Evaluate[{x[t]/. up[[1, 1]], x'[t]/. up[[1, 1]]},
Evaluate[{x[t]/. uc[[1, 1]], x'[t]/. uc[[1, 1]]}], {t, 0, 20}]

```

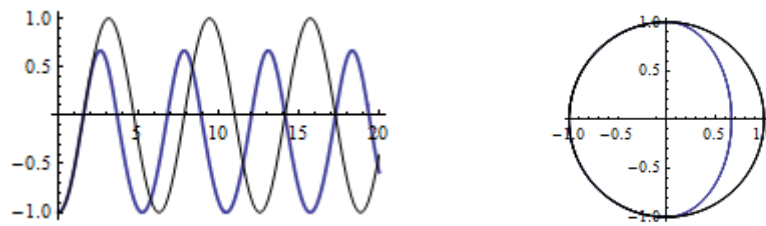



График решения показан на следующем рисунке синей (более низкой) кривой. Черная (более высокая кривая) представляет решение для случая неизменного $k=1$. На правом рисунке показана фазовая траектория синей кривой, черная (окружность) представляет фазовую траекторию для $k=1$.

Здесь для функции $k(x)$ мы выбрали представление $1+0.5 \text{ UnitStep}[x[t]]$. Допустимо использование функции Хэвисайда $1+0.5 \text{ HeavisideTheta}[x[t]]$. Можно также создать свою кусочную функцию $f[x_]=\text{Piecewise}[\{\{1, x<0\}\}, 1.5]$ и использовать ее при решении уравнения, например, так

```
up = NDSolve[{x''[t] + f[x[t]]^2 x[t] == 0,
              x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
```

В последнем случае кусков функции f можно задать много.

```
f[x_] = Piecewise[{{2, x < -1}, {1.5, x < 0}, {1, x < 1}}, 0.5];
```

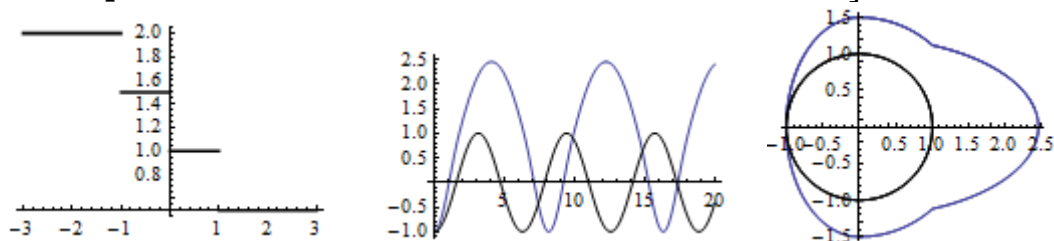
```
Plot[f[x], {x, -3, 3}]
```

```
uc = NDSolve[{x''[t] + x[t] == 0, x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
```

```
up = NDSolve[{x''[t] + f[x[t]]^2 x[t] == 0,
              x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
```

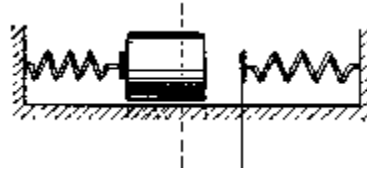
```
Plot[{x[t]/.up[[1, 1]], x[t]/.uc[[1, 1]]}, {t, 0, 20}]
```

```
ParametricPlot[{Evaluate[{x[t]/.up[[1, 1]], x'[t]/.up[[1, 1]]},
                        Evaluate[{x[t]/.uc[[1, 1]], x'[t]/.uc[[1, 1]]}], {t, 0, 20},
                AspectRatio -> Automatic, PlotPoints -> 1000]
```



На предыдущем рисунке слева показана, созданная нами кусочная функция, в середине – график решения (синяя кривая), справа – фазовая траектория (синяя кривая).

Пример 12.2. Рассмотрим систему, показанную на следующем рисунке. Груз массы m находится посередине, обе пружины касаются его, но не соединяются с ним. Когда груз находится посередине, растяжение пружин равно нулю. Жесткости пружин разные и их массой можно пренебречь. При смещении груза из положения равновесия возникают колебания.



Уравнение движения груза при $x < 0$ имеет вид

$$\ddot{x} + \frac{c_1}{m}x = 0, \quad x < 0$$

Уравнение движения груза при $x > 0$ имеет вид

$$\ddot{x} + \frac{c_2}{m}x = 0, \quad x > 0$$

Таким образом, единое уравнение можно записать в виде

$$\ddot{x} + k^2x = 0, \quad (1)$$

где

$$k^2(x) = \begin{cases} c_1/m, & x < 0 \\ c_2/m, & x > 0 \end{cases} = \begin{cases} k_1^2, & x < 0 \\ k_2^2, & x > 0 \end{cases} = k_1^2 + (k_2^2 - k_1^2)H(x)$$

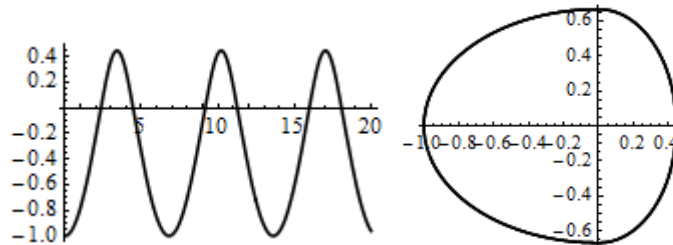
где $H(x)$ – функция Хэвисайда. Обратите внимание, что коэффициент $k(x)$ зависит от решения $x(t)$. Функция `DSolve` не может решить эту задачу, поэтому используем `NDSolve`.

k1 = $\frac{2}{3}$; **k2** = $\frac{3}{2}$; **k[x_]** = `Piecewise[{{k12, x < 0}, {k22, x ≥ 0}}]`;

s = `NDSolve[{x''[t] + k[x[t]]x[t] == 0, x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]`

`Plot[s[[1, 1, 2]][t], {t, 0, 20}]`

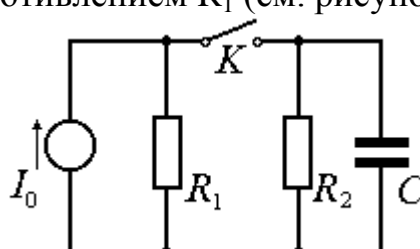
`ParametricPlot[Evaluate[{s[[1, 1, 2]][t], D[s[[1, 1, 2]][t], t]}, {t, 0, 20}]`



Слева показан график зависимости решения от времени, справа – фазовая траектория.

4.5.13 Генератор пилообразных напряжений

Пример 13.1. Простой генератор напряжений пилообразной формы состоит из параллельно соединенных емкости C и сопротивления R_2 , которые периодически подключаются к параллельно соединенным генератором постоянного тока I_0 и сопротивлением R_1 (см. рисунок).



Коммутирующее устройство, периодически осуществляющее включение и отключение, ради простоты на схеме представлено в виде рубильника К, который замкнут в течение первой части цикла длительностью t_1 секунд и разомкнут в течение второй части периода длительностью $T-t_1$ секунд. Задача состоит в определении закона нарастания напряжения между обкладками конденсатора С (в начальный момент конденсатор разряжен).

Здесь мы рассмотрим не периодический процесс, а только один шаг, состоящий во включении и отключении рубильника. Обозначим через $u_1(t)$ напряжение конденсатора при замкнутом рубильнике, т.е. в интервале времени $0=t_0 < t < t_1$, и $u_2(t)$ напряжение на конденсаторе при разомкнутом рубильнике для моментов времени $t > t_1$.

Дифференциальные уравнения для разных интервалов времени имеют вид

$$C \frac{du_1}{dt} + \frac{1}{R_0} u_1 = I_0 \text{ для } 0 \leq t \leq t_1;$$

$$C \frac{du_2}{dt} + \frac{1}{R_2} u_2 = 0 \text{ для } t \geq t_1,$$

где $\frac{1}{R_0} = \frac{1}{R_1} + \frac{1}{R_2}$. Так как напряжение на конденсаторе может меняться только непрерывно, то значение u_1 в момент t_1 равно начальному значению u_2 в этот же момент времени $u_1(t_1) = u_2(t_1)$. Оба эти уравнения мы можем представить в виде одного ДУ $\frac{du}{dt} + a(t) \cdot u = f(t)$, где $a(t)$ и $f(t)$ кусочно-постоянные функции.

На участке $0 \leq t < t_1$ функция $a(t)$ равна $a_1 = \frac{1}{C R_0}$, и для $t \geq t_1$ равна $a_2 = \frac{1}{C R_2}$.

Функция $f(t)$ на участке $0 \leq t < t_1$ равна $f_1 = \frac{I_0}{C}$, а для $t \geq t_1$ равна $f_2 = 0$.

Соответственно функция $u(t)$ на этих временных участках равна $u_1(t)$ и $u_2(t)$. Т.о. мы приходим к задаче

$$\frac{du}{dt} + a(t) \cdot u = f(t), \quad u(0) = 0,$$

где

$$a(t) = \begin{cases} a_1, & t < t_1 \\ a_2, & t > t_1 \end{cases} \quad \text{и} \quad f(t) = \begin{cases} f_1, & t < t_1 \\ 0, & t > t_1 \end{cases}$$

Теперь можно написать код решения задачи (положим $C=1$).

R1 = 1; R2 = 1; I0 = 1; f1 = I0; t1 = 1;

R0 = $\frac{R1 R2}{R1 + R2}$; a1 = $\frac{1}{R0}$; a2 = $\frac{1}{R2}$;

a[t_] = Piecewise[{{a1, t < t1}}, a2];

f[t_] = Piecewise[{{f1, t < t1}}, 0];

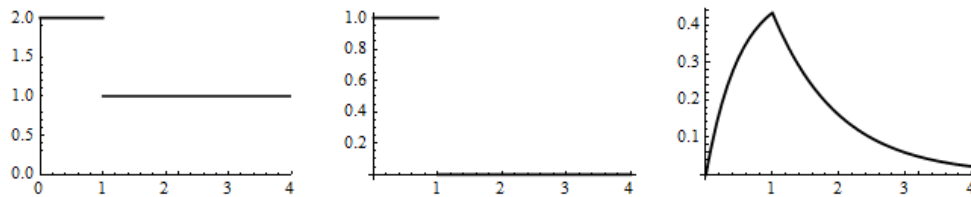
sU = DSolve[{u'[t] + a[t]u[t] == f[t], u[0] == 0}, u, t];

p1 = Plot[a[t], {t, 0, 4}, PlotRange -> {{0, 4}, {0, 2}}];

```

p2 = Plot[f[t], {t, 0, 4}];
p3 = Plot[sU[[1, 1, 2, 2]], {t, 0, 4}];
GraphicsRow[{p1, p2, p3}]

```



Слева показан график функции $a(t)$, в середине – функции $f(t)$, справа – график решения.

Пример 13.2. Пусть рубильник включается и выключается периодически. Это значит, что функции $a(t)$ и $f(t)$ должны совпадать с одноименными функциями из предыдущего примера на отрезке периода T и затем повторяться периодически. Для моделирования кусочно – постоянной периодической функции создадим одну специальную функцию, которую назовем *Periodic Step Function* = *Psf*

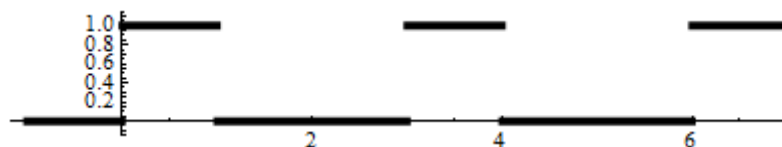
$$Psf(x, a, w) = \left\lfloor \frac{x}{w} \right\rfloor - \left\lfloor \frac{x-a}{w} \right\rfloor$$

Это периодическая с периодом w функция. Для $a < w$ на отрезке $0 \leq x < a$ функция равна 1, на остальном куске периода $a \leq x < w$ функция равна 0. Случай $a \geq w$ мы не рассматриваем. На следующем рисунке приведен график функции $Psf(x, 1, 3)$.

```

Psf[x_, a_, w_] = Floor[x/w] - Floor[(x - a)/w];
Plot[Psf[x, 1, 3], {x, -1, 7}, AspectRatio -> Automatic]

```

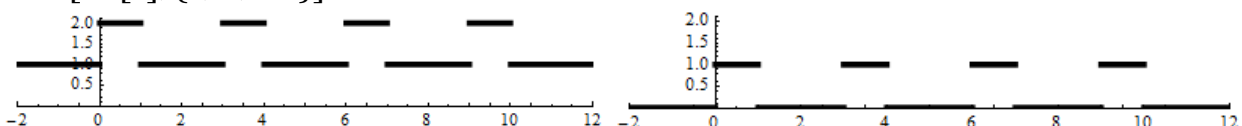


Эта импульсивная функция принимает значения +1 и 0. Используя эту функцию, можно смоделировать периодические функции, совпадающие с $a(t)$ и $f(t)$ на отрезке периода T . Пусть $T=3$. Тогда создадим требуемые функции и решим соответствующее ОДУ

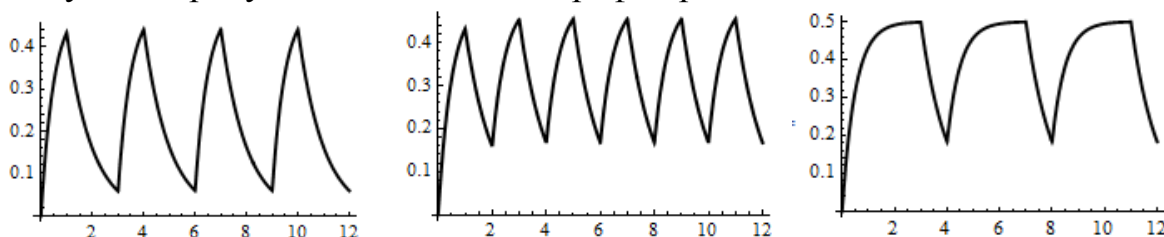
```

Remove[t, a, f, u]; t1 = 1; T = 3;
a[t_] = 1 + Psf[t, t1, T];
f[t_] = Psf[t, t1, T];
Plot[a[t], {t, -2, 12}, PlotRange -> {{-2, 12}, {0, 2.1}}]
Plot[f[t], {t, -2, 12}, PlotRange -> {{-2, 12}, {0, 2.1}}]
nds = NDSolve[{u'[t] + a[t]u[t] == f[t], u[0] == 0}, u, {t, 0, 12}]
xs = u /. nds[[1]];
Plot[xs[t], {t, 0, 12}]

```



На предыдущем рисунке слева показана периодическая кусочно – постоянная функция $a(t)$, а справа – периодическая кусочно – постоянная функция $f(t)$. На следующем рисунке слева показан график решения.



В предыдущем решении время включения рубильника составляло треть времени одного периода ($t_1 = 1; T = 3$). Если положить $t_1 = 1; T = 2$, то график решения будет таким, как показано на предыдущем рисунке в середине. Для его построения в предыдущем коде вы должны параметру T присвоить другое значение, например, $T = 2$. В случае $t_1 = 3; T = 4$ график решения получится таким, как показано на рисунке справа.

4.5.14 Двухвидовая модель Вольтерра «хищник – жертва»

Рассмотрим двухвидовую модель «хищник – жертва», впервые построенную Вольтерра для объяснения колебаний рыбных уловов. Имеются два биологических вида, численностью в момент времени t , соответственно, $x(t)$ и $y(t)$. Особи первого вида являются пищей для особей второго вида (хищников). Численности популяций в начальный момент времени известны. Требуется определить численность видов в произвольный момент времени. Математической моделью задачи является система дифференциальных уравнений Лотки – Вольтерра

$$\begin{cases} \frac{dx}{dt} = (a - b y) x \\ \frac{dy}{dt} = (-c + d x) y \end{cases}$$

где a, b, c, d – положительные константы. Проведем расчет численности популяций при $a = 3, b = 3, c = 1, d = 1$, для нескольких вариантов начальных условий: $x(0) = 2, y(0) = 1$; $x(0) = 1, y(0) = 2$; $x(0) = 1, y(0) = 3$; $x(0) = 3, y(0) = 2$. Организуем начальные условия в виде списка и построим для них фазовые траектории.

a = 3; b = 3; c = 1; d = 1;

sys := {x'[t] == (a - b y[t]) x[t], y'[t] == (-c + d x[t]) y[t]}

bc := {{x[0] == 2, y[0] == 1}, {x[0] == 1, y[0] == 2},
{x[0] == 1, y[0] == 3}, {x[0] == 3, y[0] == 2}}

col := {{Black}, {Blue}, {Green}, {Cyan}}

pp = Table[se = Join[sys, bc[[i]]];

u = NDSolve[se, {x, y}, {t, 0, 7};

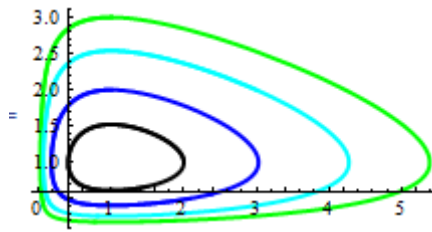
fx = u[[1, 1, 2]];

fy = u[[1, 2, 2]];

```

p = ParametricPlot[{fx[t], fy[t]}, {t, 0, 7},
  PlotStyle -> {Thickness[0.01], col[[i]]};
p, {i, 1, 4}];
Show[pp, PlotRange -> All]

```



Из вида фазовых траекторий видно, что численность популяций меняется периодически. Для более детального анализа фазовых траекторий можно использовать «манипулятор» (следующий рисунок слева). В нем мы будем менять начальные значения x_0 , y_0 искомых функций и интервал времени t_{\max} .

```

a = 3; b = 3; c = 1; d = 1;
Module[{x, y, sys, se, u, fx, fy},
  sys := {x'[t] == (3 - 3y[t])x[t], y'[t] == (-1 + x[t])y[t]};
  Manipulate[
    se = Join[sys, {x[0] == x0, y[0] == y0}];
    u = NDSolve[se, {x, y}, {t, 0, tmax}];
    ParametricPlot[Evaluate[{fx[t], fy[t]}, {t, 0, tmax},
      PlotStyle -> {Thickness[0.01], {Black}},
      PlotRange -> {{-1, 10}, {-1, 5}},
      {x0, 1, 4, 0.5}, {y0, 1, 4, 0.5}, {tmax, 0.05, 7, 0.05},
      AutoAction -> False, Initialization: >
      {fx := u[[1, 1, 2]]; fy := u[[1, 2, 2]]}]

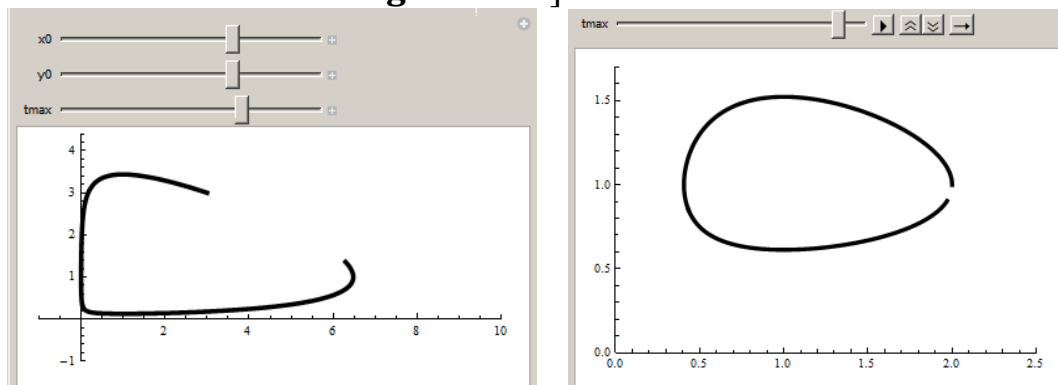
```

Можно построить анимацию (следующий рисунок справа)

```

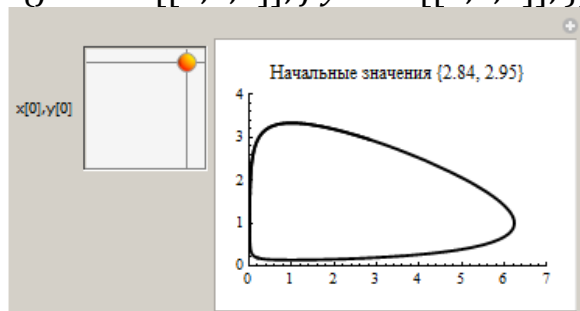
Animate[
  u = NDSolve[{x'[t] == (a - by[t])x[t], y'[t] == (-c + dx[t])y[t],
    x[0] == 2, y[0] == 1}, {x, y}, {t, 0, tmax}];
  fx = u[[1, 1, 2]];
  fy = u[[1, 2, 2]];
  ParametricPlot[Evaluate[{fx[t], fy[t]}, {t, 0, tmax},
    PlotStyle -> {Thickness[0.01], {Black}},
    PlotRange -> {{0, 2.5}, {0, 1.7}}, {tmax, 0.01, 4, 0.01},
    AnimationRunning -> False]

```



Можно использовать «2D манипулятор» для одновременного изменения начальных значений популяций.

```
Module[{u, se, x, y, pt, sys, fx, fy},
  a = 3; b = 3; c = 1; d = 1;
  sys = {x'[t] == (a - b y[t]) x[t], y'[t] == (-c + d x[t]) y[t]};
  Manipulate[
    se := Join[sys, {x[0] == pt[[1]], y[0] == pt[[2]]}];
    u = NDSolve[se, {x, y}, {t, 0, 7}];
    ParametricPlot[{fx[t], fy[t]}, {t, 0, 7},
      PlotStyle → {Thickness[0.01], Black},
      PlotRange → {{0, 7}, {0, 4}}, PlotLabel →
        StringJoin["Начальные значения", ToString[pt]],
      {{pt, {2, 2}, x[0], y[0]}, {1, 1}, {3, 3}},
      ControlPlacement → Left, AutoAction → False,
      Initialization: > {fx := u[[1, 1, 2]]; fy := u[[1, 2, 2]]; }]]
```



Весь код мы включили в функцию Module для того, чтобы основные переменные кода, стоящие в списке первого аргумента и являющиеся локальными для модуля, не взаимодействовали с одноименными переменными блоков кода других примеров. Имеет смысл коды всех примеров обрамлять этой функцией.

Замечания о выполнении примеров

Если вы начинаете новую сессию работы с новым документом, то ввод и выполнение кода любого примера пройдет гладко так, как описано нами. Все примеры протестированы в системе Mathematica 9. Если вы работаете с версией *Mathematica* 6, 7 или 8, то в них нет некоторых функций, которые мы описывали и использовали в примерах. Например, в прежних версиях системы нет функций `DifferentialRoot`, `DifferentialRootReduce`, `NDSolveValue`, `ParametricNDSolve`, `ParametricNDSolveValue` и некоторых связанных с ними функций. В примерах с функцией `NDSolveValue` вы можете выполнить замену на функцию `NDSolve`. Конечно придется выполнить и некоторые изменения последующего кода, поскольку результаты, возвращаемые этими функциями, используются по-разному. Коды, содержащие другие из указанных функций, вам придется существенно переработать. В версиях *Mathematica* 4 и 5, кроме прочего, нет управляющих элементов `Manipulate` и других. Также в новых версиях

системы некоторые опции графических функций переработаны и имеют другие названия.

Более существенные замечания касаются случая, когда в одном документе вы будете выполнять множество из приведенных нами примеров. В этом случае будет возникать конфликт имен переменных, которые, в целях краткости, мы называем одинаково x , y , t и т.д. Если перед кодом примера вы будете добавлять команду `Remove[имя1, имя2, ...]`, где `имя1`, `имя2`, это имена переменных, используемых в коде, то в большинстве случаев этого достаточно для корректного выполнения текущего примера. Однако, в таком случае выполнение текущего примера может приводить к порче результатов некоторых примеров, выполненных вами ранее. Чтобы защитить код примера от влияния других примеров, его (код) следует включить в тело функции `Module` или `DynamicModule`. Первым аргументом этих функций должен быть список имен защищаемых (локализуемых в блоке) переменных, а вторым – код примера, команды которого нужно (обязательно) разделять точкой с запятой. При этом функцию `DynamicModule` нужно использовать тогда, когда в коде примера имеются динамические переменные. С целью экономии в большинстве примеров этого пособия мы эти функции не использовали.